VM_pRAY an efficient ray tracing algorithm on a distributed memory parallel computer

Didier Badouel, Thierry Priol

INRIA

UNITE DE RECHERCHE
INRIA-RENNES

Rapports de Recherche

N° 1198

Programme 2
Structures Nouvelles d'Ordinateurs

VM_pRAY
AN EFFICIENT RAY TRACING ALGORITHM ON A DISTRIBUTED MEMORY PARALLEL COMPUTER

Didier BADOUEL
Thierry PRIOL

Mars 1990

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél (1) 39 63 55 11

RR_1198

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

VM_pRAY*
An Efficient Ray Tracing Algorithm on a Distributed Memory Parallel Computer

Didier Badouel[†]
Thierry Priol[‡]
IRISA[§]

Publication interne n°506 - Janvier 1990 - 52 pages

Abstract

The production of realistic image generated by computer requires a huge amount of computation and a large memory capacity. The use of highly parallel machines allows this process to be performed faster. Distributed memory parallel computers (DMPCs), such as hypercubes or *transputer*-based machines, offer an interesting performance/cost ratio when assuming that a load balancing and a partition of the data domain have been performed. This paper presents a software running on an iPSC/2 whose name is *VM_pRAY*. The aim of this experimental software is to show that using a virtual shared memory is an efficient strategy for parallelizing algorithms which, like ray tracing, use large read-only databases with no obvious data domain decomposition. The source code of *VM_pRAY* for an hypercube iPSC/2 is given in the appendix.

Un algorithme de lancer de rayon performant pour une machine à mémoire distribuée

Résumé

La production d'images de synthèse réalistes nécessite de nombreux calculs ainsi que l'utilisation de bases de données de taille très importantes. L'utilisation de machines massivement parallèles permet de réduire de façon significative les temps de traitement. Les machines parallèles à mémoire distribuée, telles que les hypercubes ou les machines à base de *transputers*, offrent un rapport performance/coût intéressant dès lors qu'une répartition équilibrée des calculs et des données est obtenue. Cet article présente un logiciel appelé *VM_pRAY*. Le but de ce logiciel expérimental est de montrer que l'émulation d'une mémoire partagée virtuelle est une stratégie efficace pour la parallélisation d'algorithmes, tel que le lancer de rayon, qui utilisent de grandes bases de données en lecture et dont le domaine est difficilement décomposable *a priori*. Les sources de *VM_pRAY*, pour un hypercube iPSC/2, sont fournis en annexe.

*Virtual Memory parallel RAYtracer
[†]IRISA / University of Rennes I
[‡]IRISA / INRIA
[§]IRISA is a joint laboratory of INRIA, CNRS, University of Rennes I and INSA of Rennes.
Ce travail a été accompli à l'IRISA, dans l'équipe API.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

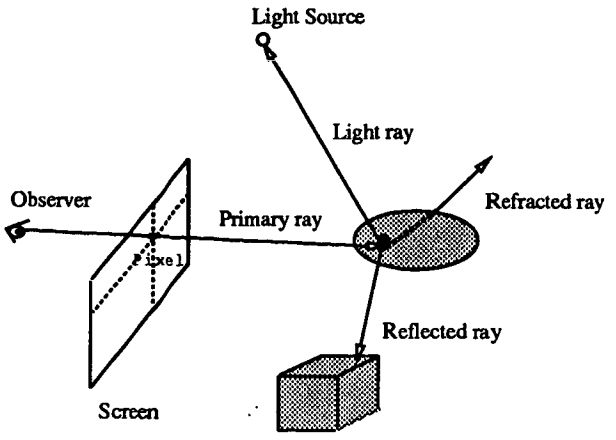

Figure 1 : The ray tracing principle.

1 Introduction

The ray tracing algorithm is used in computer graphics for rendering high quality image synthesis. It is based on simple optics' laws taking into account the effects such as shading, reflection and refraction. This graphic algorithm also is well known for its processing ressources requirement. Despite numerous improvements, the ray tracing algorithm is still too slow on sequential computers. Therefore, low cost high parallel powerful computers, such as distributed memory parallel computers (DMPCs) seem to be an opportunity for speeding up this algorithm. This paper presents *VM_pRAY*, a parallel ray tracing algorithm capable to render scenes of more than one million polygons on an hypercube iPSC/2. Section 2 presents the basic procedures used in our algorithm, while section 3 describes the use of a *virtual shared memory* for parallelizing this algorithm. Results are given in section 3.6 to show the performances of *VM_pRAY*.

2 Computation involved in the ray tracing algorithm

The ray tracing algorithm simulates the operation of a camera, following *light rays* in reverse order. The basic operation consists in tracing a ray from an *origin* point towards a *direction* in order to evaluate a light contribution. The closest intersection (*impact* point) between the ray and the scene determines the object, if one exists, which contributes to this evaluation. As shown in Figure 1, the computation of each pixel of a simulated screen plane consists in shooting a ray from an *observer* through this pixel (*primary* rays). When an impact point is found, light sources' contributions to a pixel intensity are computed by shooting rays (*light* rays) from this point to each light source to determine if the relevant point is shadowed. According to the photometric properties of the intersected objects, new rays are shot from the impact point, in order to take into account the contribution of the neighboring objects [5, 8, 15]. If the object is transparent (respectively reflective) a ray is shot in the refracted (respectively reflected) direction (*secondary* rays).

Geometric computations are used to find the closest intersection point between a ray and the objects in the scene. Their number increases with the *photometric* complexity of the scene (i.e. with the number of rays) and with the *geometric* complexity of the scene (i.e. with the number and the shape of the objects). Computing realistic images requires several millions of rays and several hundreds of thousands objects. It is this large number of ray/object intersections which makes the ray tracing a very expensive method. Several attempts have been proposed to minimize the amount of ray/object intersection. These solutions are based on what we call an *object access structure* which allows a fast search for objects along a ray path.

This section presents the basic algorithms of *VM_pRAY* dealing with geometric computations. *VM_pRAY* uses polygonal objects, and both a *regular grid* [1, 6] as an *object access structure* and an object extent called *slabs* [9].

For the basic algorithms, we use the following representation :

- The parametric representation of a ray is :

$$r(t) = O + Dt \tag{1}$$

  where, O is the origin of the ray, D the direction of the ray, and t the parameter of the representation.

- A polygon is described by its vertices V_i ($i \in \{0, \cdots, n-1\}, n \geq 3$). Let x_i, y_i and z_i the coordinates of the vertex V_i. Assuming that $\overrightarrow{V_0V_1}$ and $\overrightarrow{V_0V_2}$ are not colinear, the normal vector N of the plane containing the polygon is given by :

$$\overrightarrow{N} = \overrightarrow{V_0V_1} \times \overrightarrow{V_0V_2}$$

  For each point P of the plane the quantity $P.N$ is constant. This constant value is computed by the dot product $d = -V_0.N$. The implicit representation of the plane,

$$N.P + d = 0 \tag{2}$$

  is computed once for all, and stored in the polygon data structure.

2.1 Regular 3D grid

The rectangular scene extent is subdivided into a 3D grid whose elements are voxels. With each is associated a list of polygons intersecting it. For each ray, the grid provides a list of polygons whose location is close to the ray direction. To avoid repeated intersections between a polygon shared by severals voxels and a given ray, an identifier (id_{ray}) is stored in the polygon data structure and represents the last ray checked for an intersection with this polygon.

The grid traversal method is the one described in [1]. Beforehand, the first voxel encountered by the ray is determined. This voxel is either the voxel containing the origin of the ray (O) or the entry voxel when the ray comes from outside the grid. For each ray traversing the grid, the following values are initialized :

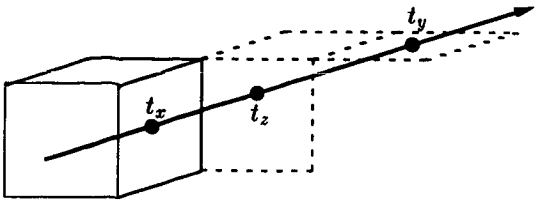

Figure 2 : Traveling a 3D grid.

- the constants δt_x, δt_y and δt_z represent the increment of t in each direction X, Y, Z.
- the variables t_x, t_y and t_z represent the values of t corresponding to the next voxel boundary in the X, Y or Z direction (see Figure 2).

An incremental step through the grid is done using :

if $t_x < t_y$ and $t_x < t_z$
then $x \leftarrow x + 1$;
$t_x \leftarrow t_x + \delta t_x$;
else **if** $t_y < t_z$
then $y \leftarrow y + 1$;
$t_y \leftarrow t_y + \delta t_y$;
else $z \leftarrow z + 1$;
$t_z \leftarrow t_z + \delta t_z$;

2.2 Slabs

Before computing the ray/polygon intersection, a test is made using the *slabs* (cf [9]). The slabs are convex extents delimited by pairs of parallel planes (see in figure 3 a 2D example). One slab is characterized by a normal direction N_i and two values d_i^{min} and d_i^{max}, so that the equation of the planes enclosing a polygon in the direction N_i are :

$$N_i \cdot P + d_i^{min} = 0 \quad \text{and} \quad N_i \cdot P + d_i^{max} = 0 \tag{3}$$

d_i^{min} and d_i^{max} are evaluated by projecting each vertex V_j according to direction N_i :

$$d_{ij} = N_i \cdot V_j \qquad d_i^{min} = \min_j(d_{ij}) \qquad d_i^{max} = \max_j(d_{ij})$$

The values d_i^{min} and d_i^{max} are stored in the polygon data structure. During the synthesis task, the ray/slab intersection results in an interval $[t_i^{min}, t_i^{max}]$. These values are computed using the ray representation (Equ. 1) and the slabs representation (Equ. 3) :

$$t_i^{min} = \frac{d_i^{min} - N_i.O}{N_i.D} \qquad t_i^{max} = \frac{d_i^{max} - N_i.O}{N_i.D}$$

For each ray, the following values are pre-computed :

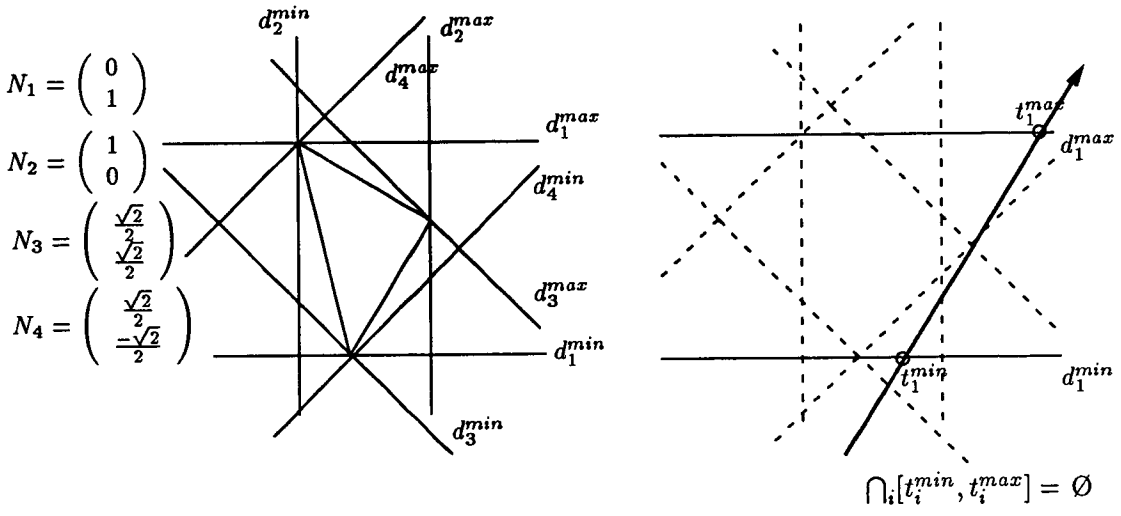

Figure 3 : Description and intersection of the *slabs* extent.

$$S_i = N_i \cdot O \quad \text{and} \quad T_i = \frac{1}{N_i \cdot D}$$

Therefore, a ray/slab intersection only requires the following computations :

$$t_i^{min} = (d_i^{min} - S_i)T_i \quad \text{and} \quad t_i^{max} = (d_i^{max} - S_i)T_i$$

Furthermore, as soon as $\bigcap_i [t_i^{min}, t_i^{max}]$ is empty, the ray does not intersect the object since it does not intersect its extent.

2.3 Ray/polygon intersection

The following ray/polygon algorithm is quite similar but faster to the one described in [14]. This algorithm consists of two steps :

- find the intersection point P between the ray and the polygon plane.
- test if P lies inside the polygon.

The parameter t corresponding to the ray/plane intersection point can be obtained by using equations (1) and (2) :

$$t = -\frac{d + N.O}{N.D} \tag{4}$$

Three tests are made for this t value :

- If polygon and ray are parallel ($N.D = 0$), the intersection is rejected.
- If the intersection is behind the origin of the ray ($t \leq 0$), the intersection is rejected.

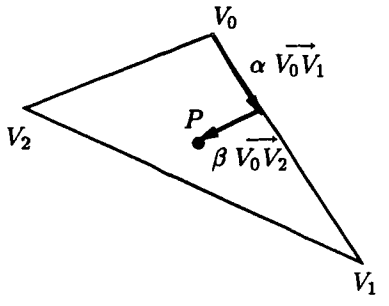

Figure 4 : Parametric representation of the point P.

- If a closer intersection has been already found for the ray ($t > t_{ray}$), the intersection is rejected.

Then, we must determine if the intersection point is inside the polygon. This algorithm is applied to triangles but can be easily extended to convex polygon. To efficiently compute a ray/triangle intersection, the triangle is parametrized by two values α and β. Indeed, the point P within the triangle plane verifies :

$$\overrightarrow{V_0P} = \alpha.\ \overrightarrow{V_0V_1} + \beta.\ \overrightarrow{V_0V_2} \tag{5}$$

The point P is inside the triangle $(V_0V_1V_2)$ if :

$$\alpha \geq 0, \beta \geq 0 \text{ and } \alpha + \beta \leq 1$$

The computation of α, β requires to solve a system of three equations of two unknowns which can be reduced to a system of two equations of two unknowns when working in a plane, with two-dimensional coordinates. We wish to project the polygon onto one of the primary planes, either XY, XZ, or YZ. If the polygon is exactly perpendicular to one of these planes, its projection onto that plane will be a single line. To avoid this problem, and to make sure that the projection is as large as possible, we find the dominant axis of the normal vector and use the plane perpendicular to that axis. As in [14], we compute the value i_0,

$$i_0 = \begin{cases} 0 & if\ |N_x| = Max(|N_x|, |N_y|, |N_z|). \\ 1 & if\ |N_y| = Max(|N_x|, |N_y|, |N_z|). \\ 2 & if\ |N_z| = Max(|N_x|, |N_y|, |N_z|). \end{cases}$$

Let us onsider i_1 and i_2 (i_1 and $i_2 \in \{0, 1, 2\}$), the indices (different from i_0) representing the two other components, and (u, v) the two-dimensional coordinates of the vectors $\overrightarrow{V_0P}$, $\overrightarrow{V_0V_1}$ and $\overrightarrow{V_0V_2}$:

$$u_0 = P_{i_1} - V_{0_{i_1}} \qquad u_1 = V_{1_{i_1}} - V_{0_{i_1}} \qquad u_2 = V_{2_{i_1}} - V_{0_{i_1}}$$

$$v_0 = P_{i_2} - V_{0_{i_2}} \qquad v_1 = V_{1_{i_2}} - V_{0_{i_2}} \qquad v_2 = V_{2_{i_2}} - V_{0_{i_2}}$$

Then, the solutions of the system are :

$$\alpha = \frac{det\begin{pmatrix} u_0 & u_2 \\ v_0 & v_2 \end{pmatrix}}{det\begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix}} \quad \text{and} \quad \beta = \frac{det\begin{pmatrix} u_1 & u_0 \\ v_1 & v_0 \end{pmatrix}}{det\begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix}}$$

The parameters α and β can be used to compute the interpolated normal vector at the point P, and can be also used to compute the entry of a texture map. The interpolated normal vector N_P at the point P may be obtained by :

$$N_P = (1 - (\alpha + \beta)).N_0 + \alpha.N_1 + \beta.N_2$$

2.4 Partitioning the ray tracing algorithm

Using polygons and 3D grid, the geometrical data structures involved in *VM_pRAY* rapidly reaches several tens millions of bytes (Mbytes). In our results, we present a database which requires 140 Mbytes of memory. This problem of memory amount becomes even more crucial when using texture databases. Thus, our study of parallelization did not hold the algorithms based on processing without dataflow as they do not achieve a data distribution which allow the rendering of complex scenes.

Since DMPCs are not fine grain parallel computers, the basic computation involved in *VM_pRAY* is the evaluation of one pixel. The computation can be easily distributed among the processors, as the evaluation of each pixel is independent of the others. Since there are much more pixels than processors, load balancing can be achieve by using a server/client programming model. A server process assigns the computation of a pixel to a client process running on a non-busy processor.

The problem of parallelizing an algorithm is to efficiently insure, at once, a data domain decomposition and a computation partition. We have yet experimented a parallel ray tracing algorithm [11, 12] based on processing with ray dataflow. This algorithm took up the Cleary's idea [4] and subdivides the scene extent into sub-regions distributed among the differents processor elements (PEs). Rays are exchanged between PEs when they move from region to region. This experience has provided several interesting solutions for insuring a static load balancing, but the efficiency of this algorithm rapidly decreases when the number of PEs increases. The reason of this behavior is due to the message traffic increase.

Our first experience shows that to insure both a load balancing and a data domain decomposition for the ray tracing algorithm is not efficient when using a message based programming model. The next section presents another approach which uses a shared memory programming model to achieve a data domain decomposition. The message based programming model is just used for distributing the calculus.

Cache Device	Memory Device	Item of transfer
Registers	Main Memory	Word
Hardware Cache	Main Memory	Block
Main Memory	Secondary Storage	Page
Node Memory	Distributed Memory	Object(s)

Figure 5 : Several uses of the memory cache mechanism.

3 Emulating a read-only shared memory for ray tracing

3.1 A shared memory model of programming for DMPCs

To overcome the difficulties of the message passing programming model, several studies have tried to define mechanisms for implementing a shared data model in distributed systems [2, 3, 10]. The goal of this work is to provide a better abstraction of mapping data on a set of distributed memories. In [2] and [10], strategies for maintaining data consistency between copies of modified variables are studied in order to offer a general tool. The data management following this abstraction is quite attractive but our first objective is always the efficiency of data accesses. The aim here is to show that the emulation of a shared memory on a DMPC is the best way to parallelize algorithms such as ray tracing which use large read-only databases with no obvious data domain decomposition. With a DMPC, a part of each node's memory can be used to store a part of the shared database and the remaining part is used as a cache to speed up slow global accesses. The notion of cache, managed by software in our case, is the core of an efficient shared memory emulation.

Caches were introduced to palliate the gap between fast processor cycle times and slow large memory access times. Generally speaking, a memory cache is any hardware or software device storing in a relatively small but fast access area a selected part of a database stored in a larger but slower access memory (see Figure 5). A general presentation of cache memories can be found in [13]. The use of a cache device improves the bandwidth between the processor and its memory. In our case, it increases the bandwidth between a node and the distributed global memory, called *virtual shared memory.*

3.2 A model well suited for ray tracing

Various characteristics of the ray tracing algorithm led us to design a software tool to emulate an access to a global memory in the context of DMPCs. These characteristics are as follows:

- the huge amount of memory necessary for this algorithm makes the database load balancing as important as the computation load balancing. Increasing size problems are a challenge for DMPCs ;
- due to the coherence property and topological property of 3D objects, only a small part of the whole database is required at a given time. Thus a caching mechanism can be efficient for our problem ;

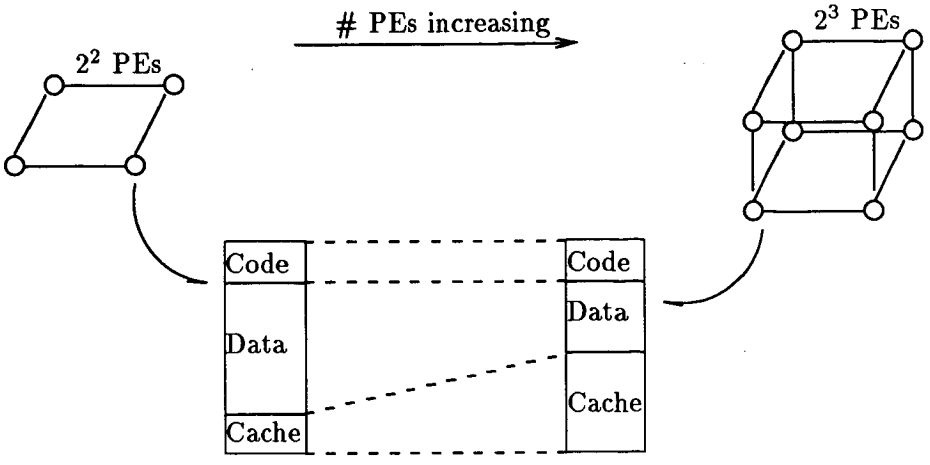

Figure 6 : A user node memory description.

- due to illumination effects (shading, reflection, refraction), the small part of database necessary to evaluate one pixel is nearly impossible to determine statically. Thus the database memory management must be dynamic ;
- the computation of an image uses the database in a read-only way, therefore the problem of data coherence management is not posed.

3.3 Implementing a virtual shared memory

In the ray tracing algorithm, the virtual shared memory contains the database and the frame buffer. The sharing of pixels will be discussed in the next section. The database contains the photometric and geometric parameters of the objects of the scene, together with the object access structure. The mechanism used to manage the virtual shared memory is called *Object Paging* where an *object* (a polygon, a voxel of the grid ... etc) is an item of a page which can be transferred between local memories. An object belongs to one and only one page, and thus its memory location is contiguous.

In our algorithm, the whole database is first equally distributed over the set of nodes without any particular strategy. Therefore each PE's memory almost contains the same number of pages. A local memory of a PE is organized as shown in figure 6. Each local memory is divided in three parts: the process code, a part of the database, and the cache memory. The two last parts are divided into pages to allow memory management.

3.4 Accessing an object in the virtual shared memory

During the synthesis task, the application can potentially access the whole database through a software memory management. For each node, when a cache miss is detected, i.e. the page is neither in its local database nor in the cache memory, then a request is sent to the node responsible for this page. When the node receive the page, it stores it in

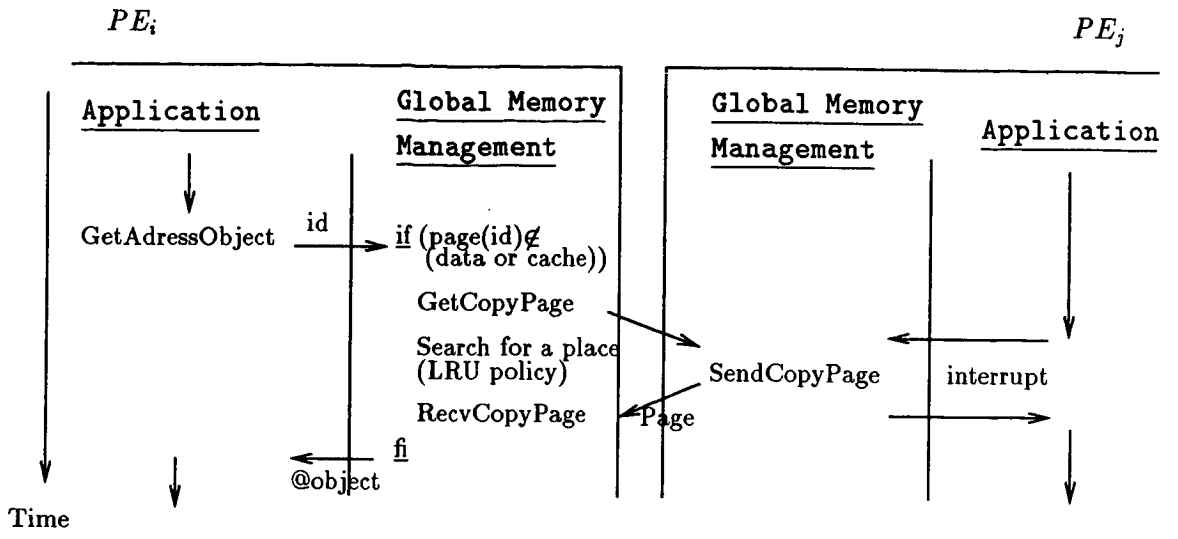

Figure 7 : Accessing a global object.

its cache memory according to a LRU (Last Recently Used) policy. This search is done during the communication of the new page, and thus causes no extra cost (see Figure 7).

In *VM_pRAY*, an *object* of the global database is characterized by two numbers : id_1 is the identifier of the class which the object belongs to, and id_2 is the member identifier within this class. The numbers (id_1,id_2) represent one unique location in the global memory. A class is a set of objects having the same type. All the objects of one class are stored in contiguous pages to make the global memory management easier. The informations relative to one class are :

- $first_{page}(id)$, the first page where the objects of class id are located.
- $size_{object}(id)$, the size of an object of class id.
- $nb_{object\_per\_page}(id)$, the number of objects of class id in one page.

Thus, in order to access a global object, we must determine :

- the page where the object is located :

$$num_{page} = first_{page}(id_1) + id_2 \ / \ nb_{object\_per\_page}(id_1)$$

- the node where the page is located :

$$num_{node} = num_{page} \ \% \ nb_{node}$$

- the page with respect to this node ?

$$dep_{node} = num_{page} \ / \ nb_{node}$$

- the object with respect to this page ?

$$dep_{page} = id_2 \ \% \ nb_{object\_per\_page}(id_1)$$

After getting the page from an other node if necessary, the address of the object is evaluated as follows :

$$@object = @page + dep_{page} \times size_{object}(id_1)$$

For better performances, we have chosen the values as power of two. Thus, all the operations necessary to calculate the address of an object only require logical operations.

3.5 Work and bitmap distribution

With a shared database, to ensure a load balancing is quite simple. Each PE is owner of a part of the bitmap. For example, if we use 32 PEs to compute an image with a 512 x 512 resolution, each PE manages a 32 x 32 sub-bitmap. We use a square (or nearly square) sub-bitmap in order to exploit as much as possible the ray coherence property. If the PEs could directly addressing the frame buffer, a centralized control would not be necessary. As we do not have this facility on the iPSC/2, the host computer insures a global management of the different sub-bitmaps in order to fill in the frame buffer. The synthesis of each sub-bitmap requires frequent global data accesses at the beginning of the task, and the number of external requests progressively decreases as the memory cache keeps the pertinent items of the global database.

When a PE completes the computation of its sub-bitmap, it sends a request to get a *work item* (i.e a set of pixels) from a PE still working on its own sub-bitmap. This request moves along a ring topology. If this request goes back without satisfaction, the PE knows that the image computation is achieved. This local termination detection is sufficient for our application.

In order to insure a load balancing, the only parameter to be determined is the size of this *work item*. If its size is minimal (i.e. work item = one pixel), then we have the best load balancing we can obtain, assuming that the computation of one pixel is indivisible over the set of nodes, but the communication cost is then high. We must not generate more work in communication activity than in computation to insure the load balancing . Experimental results (see Figure 8) show that a size of about 3 x 3 pixels offers a good compromise.

3.6 Results

Tests of our parallel ray tracing has been performed on a set of scenes call *Standard Procedural Databases* (SPD) provide by Eric Haines [7] and other scenes including the

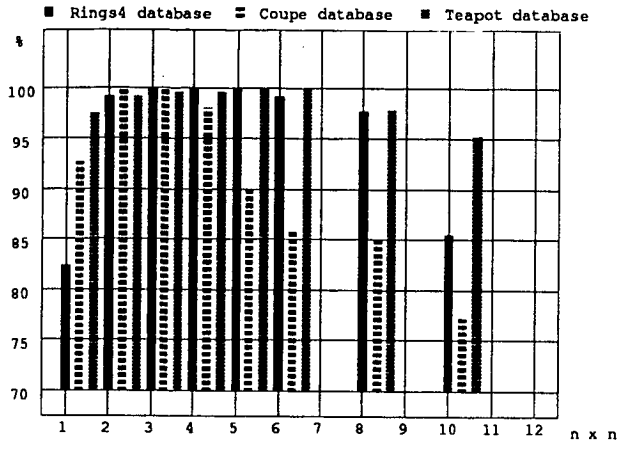

Figure 8 : Relative efficiency using different size for a *work item.*

famous *Teapot* from the university of Utah. These files are described with the Neutral File Format (NFF) of Eric Haines. Several synthesis times are given by the table in figure 11.

Figures 9 and 10 show the speedup and the efficiency obtained with this algorithm. If we compare the results obtained by this method with our previous work [11, 12], we can emphasize on the improvements brought by the shared database model of programming. The behavior of this algorithm is what a user of parallel machines expects : the use of more PEs allows to solve problems faster, and to consider larger problems. This is due to the characteristics of the software global memory management ;

- for a sufficient size of memory cache, the PEs can work efficiently since the number of requests to others is small ;
- the size of the memory cache is flexible. Indeed, with a memory fixed-sized problem, i.e. a fixed-size database, using more PEs increases the computation power of course, but also provides a better memory management as the local cache memory increases (see Figure 6).

One of our goals is to compute as large a database is possible. At present, we have rendered the *tetra10* database which contains more than one million (1 048 576) polygons. The size of this scene with its *object access structure* requires the use of 109 452 pages ($\times$ 1 280 Bytes), which represents a shared memory of about 140 MBytes. The synthesis time with 64 nodes is 8 mn 46 sec. Note that this database cannot be rendered with 32 nodes (with 4 MBytes of memory per node).

4 Conclusion

The aim of our study to parallelize the ray tracing method is to bring out a model of parallel programming well suited for this kind of algorithm. Due to the difficulty to appreciate the performance of the various parallel ray tracing algorithms, we have done and keep on

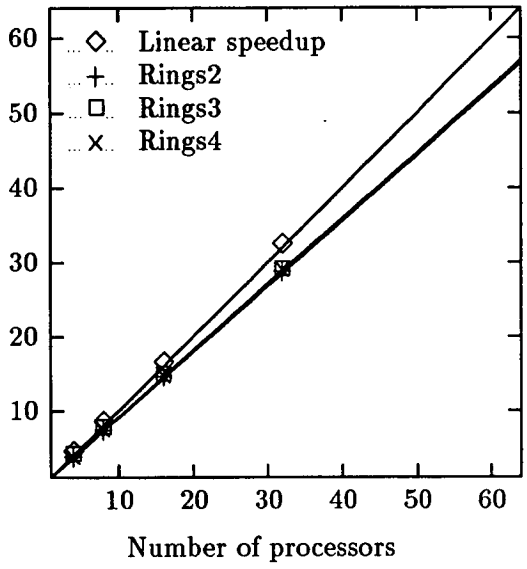

Figure 9 : Speedup for the *Rings* images.

# Proc. :	1	2	4	8	16	32	64
Rings2	1.00	0.95	0.93	0.91	0.91	0.90	0.89
Rings3	1.00	0.95	0.92	0.91	0.90	0.89	0.88
Rings4	1.00	0.95	0.93	0.92	0.91	0.90	0.88

Figure 10 : Efficiency for the *Rings* images.

Database	# Polygons	Synthesis Time
Teapot	3754	1 mn 59 sec
Coupe	15408	5 mn 44 sec
Rings4	18002	8 mn 48 sec
Tetra9	262144	2 mn 21 sec
Tetra10	1048576	8 mn 46 sec

Figure 11 : Examples of synthesis times with 64 nodes and a resolution of 512×512 pixels.

doing experiences on an iPSC/2 hypercube. Comparing the behavior of our first algorithm [11, 12] using a message passing model of programming with the behavior of the one described in this paper, which uses a shared database model of programming, we advocate the shared model approach when using large read-only database with no obvious data domain decomposition.

Bibliographie

[1] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *EUROGRAPHICS'87*, pages 3–9, Amsterdam, 1987.

[2] R. Bisiani, A. Nowatzyk, and M. Ravishankar. *Coherent Shared Memory on a Message Passing Machine.* Technical Report CMU-CS-88-204, Carnegie Mellon, December 1988.

[3] N. Carreiro and D. Gelernter. The s/net's linda kernel. *ACM Transactions Computer Systems*, May 1986.

[4] J.G Cleary, B. Wyvill, G.M. Birtwistle, and R. Vatti. *Multiprocessor Ray Tracing.* Research Report 83/128/17, University of Calgary, October 1983.

[5] R.L. Cook and K.E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1):7–24, January 1982.

[6] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: accelerated ray_tracing system. *IEEE Computer Graphics and Applications*, 16–26, April 1986.

[7] E. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 7(11):3–5, November 1987.

[8] A. Roy Hall and Donald P. Greenberg. A testbed for realistic image synthesis. *IEEE Computer Graphics and Applications*, 3(8):10–20, November 1983.

[9] T.L. Kay and J.T. Kajiya. Ray tracing complex scenes. *ACM Computer Graphics*, 20(4):269–278, August 1986.

[10] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. In *Symposium on Principles of Distributed Computing*, pages 229–239, ACM SIGACT-SIGOPS, Calgary, Canada, 1986.

[11] T. Priol. *Lancer de rayon sur des architectures parallèles: Etude et mise en œuvre.* PhD thesis, Institut de Formation Supérieure en Informatique et Communication, Rennes, juin 1989.

[12] T. Priol and K. Bouatouch. Static load balancing for a parallel ray tracing on a MIMD hypercube. *Visual Computer*, 5:109–119, March 1989.

[13] A.J. Smith. Cache memories. *Computing Surveys*, 14(3):473–530, September 1982.

[14] J.M. Snyder and A.H. Barr. Ray tracing complex models containing surface tessellations. *ACM Computer Graphics*, 21(4), July 1987.

[15] T. Whitted. An improved illmination model for shaded display. *Computer Graphics and Image Processing*, 23(6), June 1980.

A The iPSC/2 hypercube

For the *VM_pRAY* design, we have used an iPSC/2. The iPSC/2 system consists of a *cube* connected to a *host* processor. The *cube* houses all the nodes connected through a hypercube network topology. Each node consists of an Intel 80386 microprocessor supplied with a 80387 floating point co-processor, and 4 Mbytes of local memory. It is equipped with the Direct Connect Module (DCM) for high speed message routing between nodes. The performance of a 64 nodes is approximatively 256 MIPS and 20 MFLOPS. The node supports a vector extension board with peak performance 20 MFLOPS per node. The system available at IRISA is configured in 64 nodes with no vector extension. The *host* processor contains the software development tools. It is connected via a special link to node cube 0. It performs compiling, program loading and I/O operation with the hypercube. The iPSC/2 can be programmed using C or FORTRAN. A communication library handles message communication between nodes.

B A brief introduction to the C Communication library

B.1 crecv

$crecv(type, buf, len)$
Receive a message (blocking mode)

B.2 ginv

$j = ginv(i)$
Returns the position of an element in the binary-reflected Gray code sequence.

B.3 gray

$i = gray(j)$
Returns the binary-reflected Gray code for an integer.

B.4 hrecv

$hrecv(typesel, buf, len, proc)$
post a request to receive a message into a buffer, and when receipt of the message is complete, immediately execute a receive trap handler procedure. The name of the handler is the *proc* parameter.

B.5 infocount

$msglen = infocount()$
Get length of the last message.

B.6 isend

$id = isend(type, buf, len, node, pid)$
Send a message (no blocking mode)

B.7 irecv

$id = irecv(type, buf, len)$
Receive a message (no blocking mode)

B.8 led

$led(1)$ switch on the user led.
$led(0)$ switch off the user led.

B.9 masktrap

$newstate = masktrap(oldstate)$
Enable or disable a receive trap. Used for critical processing.

B.10 msgdone

$msgdone(id)$
Determine whether an isend or irecv operation is complete. (no blocking mode)

B.11 myhost

$myhost = myhost()$
Obtain the ID of the host machine.

B.12 mynode

$mynode = mynode()$
The node ID (0 to 127) of the process that initiated the routine is returned.

B.13 nodedim

$dim = nodedim()$
Delivers an integer from 0 to 7 representing the dimension of the cube.

B.14 numnodes

$num = numnodes()$
Delivers an integer from 1 to 128 representing the number of nodes in a cube.
$numnodes() == 2^{nodedim()}$

C Source files

This appendix contains a *man page* and the source code for *VM_pRAY*. The list of the source files are presented as follows :

- **Makefile** for the host code.
- **host_main.c**
- **host_comm.c**
- **host_data.c**
- **host_out.c**
- **host_page.c**
- **host_seads.c**
- **host_extern.h**
- **Makefile** for the node code.
- **node_main.c**
- **node_data.c**
- **node_comm.c**
- **node_page.c**
- **node_param.c**
- **node_photo.c**
- **node_pixel.c**
- **node_plane.c**
- **node_pol.c**
- **node_seads.c**
- **node_slabs.c**
- **node_extern.h**
- **host_nff.y**
- **ray_vec.h**
- **ray_page.h**
- **ray_comm.h**
- **ray_type.h**

NAME

VM_pRAY (Virtual Memory parallel RAYtracer)

a parallel ray tracing using polygonal databases.

SYNOPSIS

pray [**–i** *inputfile*] [**–o** *outputfile*] [**–x**] [**–v**][**–r res**][**–k coeff**][**–?**]

DESCRIPTION

pray is the version of *VM_pRAY* running on an iPSC/2.

This program reads Eric Haines' NFF file format files as input. Only polygonal objects (list of triangles) are taken into account. In order to load the database during the reading of the input file, we have added some informations in the NFF format. These informations are :

The number of polygones in the scene.
"n" (or "number") nb
The scene extent.
"g" (or "grid") xmin xmax ymin ymax zmin zmax

The viewing format is now:

v
from %g %g %g
at %g %g %g
up %g %g %g
angle %g
hither %g
resolution %d %d
number %d

and the scene extent:

grid %g %g %g %g %g %g

OPTIONS

–i file.nff
to specify a database input file. If no input file is specified, the program reads from standard input.

–o file.rvb
to specify an output file for the bitmap (24 bits depth). If no output file is specified, the bitmap is not saved.

–x to open an X window for displaying a grey bitmap (8 bits depth) in real time during synthesis. The user must specify an X color screen (8 bits depth) with the environment variable DISPLAY.

–v to make a fast synthesis of the image. The depth of the ray tree is fixed to 1. No shadowing, reflection, refraction effects are rendered.

–r res to change de resolution of the NFF input file. With this option, the image will be calculated with a *resXres* resolution.

–k coeff
to change de coefficient for the grid construction. The number of voxels in the grid will be *coeff*number_of_polygons*. The default value for *coeff* is 10.

–? to display the options.

SEE ALSO

NFF(1)

AUTHOR

Didier Badouel (e-mail : badouel@irisa.fr)

IRISA / University of Rennes I - France

WARNINGS

On account of using NFF text file format, reading the input file is very slow. Another version of VM_pRAY uses a binary input file, but requires a filter program. We think that for experimental use, NFF as direct input file format is more suitable.

This software does not use I/O node system because our iPSC/2 configuration has not I/O nodes !

ACKNOWLEDGEMENT

Many thanks go to Thierry Priol, Bruno Arnaldi and Jean-Luc Corre who contributed to this software. And special thanks to Eric Haines (NFF's author) and Mark VandeWettering (author of MTV ray tracer) who first distributed their work in the field of ray tracing.

VM_pRAY (Virtual Memory parallel RAYtracer)

```
# Makefile for VM_pRAY -
# Generate code for an iPSC/2 with 4MBytes per nodes
# If your configuration is different, you should modify
# the value of PAGENUMB in the file ray_page.h.
#
PROG=pray
CC=gcc
CFLAGS=-O
YFLAGS=-d
#
# If you use X11 Window System ....
#
#XH = -DV_X11 -I/usr/local/X11R3/include
#XL = -L/usr/local/X11R3/lib -lX11 -L/usr/local/lib
#
# .... else
#
XH =
XL =

CSRC=host_comm.c host_data.c host_main.c \
     host_out.c host_page.c host_seads.c \
     lex.c keyword.c
COBJ=host_comm.o host_data.o host_main.o \
     host_out.o host_page.o host_seads.o \
     lex.o keyword.o
OSRC=host_nff.y
OOBJ=host_nff.o

LIBS=-host -lm $(XL)

all:
        (cd ../node; make);
        make $(PROG)

$(PROG):$(COBJ) $(OOBJ)
        $(CC) $(CFLAGS) -o $(PROG) $(COBJ) $(OOBJ) $(LIBS)

host_out.o : host_out.c
        $(CC) $(CFLAGS) -c host_out.c $(XH)

allclean:
        (cd ../node; make clean);
        make clean

clean:
        rm -f $(COBJ) $(OOBJ)
        rm -f core tags

lint:
        lint $(CSRC)
```



```
/*_____________________________________________________________________

  File          : host_main.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________
  Nothing ever becomes real until it is experienced.
                                                        J. Keats        */
#include <stdio.h>
#include <sys/types.h>
#include <sys/times.h>
#include "ray_type.h"
#include "ray_comm.h"
#include "host_extern.h"

extern  time_t  time();
extern  char    *optarg;
extern  int     optind;

char    *decon = "DECON";
int     c, res = 0;
long    cloq0,cloq1;
char    *infilename = "stdin" ;
char    *deconfilename = "../nff/spirale.nff" ;
char    *outfilename = "out.rvb";

main(argc, argv)
     int argc ;
     char * argv[] ;
{
    setpid(HOST_PID);
    while ((c = getopt(argc, argv, "o:i:r:k:s:vx")) != EOF) {
      switch (c) {
      case 'o':
        outfilename = optarg;
        outF = VRAI;
        if ((fdrvb = fopen(outfilename, "w")) == NULL) {
          fprintf(stderr, "cannot open %s\t\n", outfilename);
          EXIT(-1);
        }
        break;
      case 'i':
        infilename = optarg;
        break;
      case 'r':
        res = atoi(optarg);
        if (res < 0)    res = 0;
        break;
      case 'k':
        coeff = atoi(optarg);
        break;
      case 's':
        seq = atoi(optarg);
        if (seq <= 0)   seq = 1;
        break;
      case 'v':
        vite = VRAI;
        profondeur = 1;
        break;
      case 'x':
        outX = VRAI;
        break;
```

```
      case '?':
        fprintf(stderr,"usage: %s [-i file] [-o file] ", argv[0]) ;
        fprintf(stderr,"[-r number] [-k number] [-v] [-x]\t\n");
        EXIT(-1);
      }
    }
    if (getenv(decon)!=NULL) {
      infilename = deconfilename;
    }
    system("cat ../COPYING");
    fprintf(stderr,"Loading code...\t\n");
    InitComm();
    fprintf(stderr,"Reading input file...\t\n");
    ReadSceneFile(infilename) ;
    if (res != 0)       { xres = yres = res; }
    fprintf(stderr,"\tResolution %dX%d\t\t\n",xres, yres);
    fprintf(stderr,"Loading grid...\t\t\n");
    CommVox ();

    fprintf(stderr,"Init Bitmap ...\t\t\t\n");
    init_out();

    fprintf(stderr,"Synthesis ...\t\n");

    if (outX) {
      fclose(stdin);
      (void)fopen("/dev/tty","r");
      fprintf(stderr,"GO ?\t\n");
      (void)getchar();
    }
    time(&cloq0);
    do_comm();
    time(&cloq1);
    fprintf(stdout,"Synthesis time on nodes %d sec. \t\n",cloq1 - cloq0);
    killcube(-1, -1);
    image_out ();
}
```



```
/*-------------------------------------------------------------------------
  File          : host_comm.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  -------------------------------------------------------------------------
  Don't let your mouth write no check that your tail can't cash.
                                                          Bo Diddley        */
#include <stdio.h>
#include <math.h>
#include "ray_type.h"
#include "ray_comm.h"
#include "host_extern.h"

char *malloc ();
extern char *decon;
#define MAX(A,B)          ((A) >= (B) ? (A) : (B))
#define MALLOC_N(N,T)     (T *)malloc((unsigned)(N * sizeof(T)))

MSG_p_init  pho;
MSG_pack pack;
MSG_coul msg_coul;
long node;

InitComm ()
{
  /*
   * Loading the node code.
   */
  if (getenv(decon)==NULL) {
    load ("../node/node", ALL_NODES, NODE_PID);
  }
  csend (int_TYPE, &seq, sizeof(int), ALL_NODES, NODE_PID);
  poly_page = CreateObj(sizeof(Poly));
  t_pol = MALLOC_N(poly_page, Poly);
  Tvoxlist = MALLOC_N(maxvox, Voxlist);
  if ((t_pol == NULL) || (Tvoxlist == NULL))    {
    fprintf(stderr, "cannot alloc enough memory.\n");
    EXIT(-1);
  }
  OpenPage();
}

CommVox ()
{
  long  vox_node,i,nb;
  /*
   * Loading GRID and VOXELS.
   */
  SendObj((char *)Tvoxlist, sizeof(Voxlist), ind_Tvox);
  SendObj((char *)seads, sizeof(long), size_seads);
  ClosePage();
  csend (long_TYPE, &n_pol, sizeof(long), ALL_NODES, NODE_PID);
  /*
   * Loading the SEADS parameters.
   */
  csend (int_TYPE, &nb_cell_x, sizeof(int), -1, NODE_PID);
  csend (int_TYPE, &nb_cell_y, sizeof(int), -1, NODE_PID);
  csend (int_TYPE, &nb_cell_z, sizeof(int), -1, NODE_PID);
  csend (Flt_TYPE, &dx_vox, sizeof(Flt), -1, NODE_PID);
  csend (Flt_TYPE, &dy_vox, sizeof(Flt), -1, NODE_PID);
  csend (Flt_TYPE, &dz_vox, sizeof(Flt), -1, NODE_PID);
```

```
  csend (Flt_TYPE, &x_min, sizeof(Flt), -1, NODE_PID);
  csend (Flt_TYPE, &x_max, sizeof(Flt), -1, NODE_PID);
  csend (Flt_TYPE, &y_min, sizeof(Flt), -1, NODE_PID);
  csend (Flt_TYPE, &y_max, sizeof(Flt), -1, NODE_PID);
  csend (Flt_TYPE, &z_min, sizeof(Flt), -1, NODE_PID);
  csend (Flt_TYPE, &z_max, sizeof(Flt), -1, NODE_PID);
  /*
   * Loading the photometric parameters.
   */
  pho.xres  = xres;
  pho.yres  = yres;
  pho.vite  = vite;
  pho.n_lum = n_lum;
  pho.n_pho = n_pho;
  pho.n_pol = n_pol;
  pho.profondeur = profondeur;
  bcopy ((char*)fond,(char*)pho.fond, sizeof(Color));
  bcopy ((char*)ambiant,(char*)pho.ambiant, sizeof(Color));
  bcopy ((char*)&vue,(char*)&pho.vue, sizeof(Vue));
  csend (P_INIT_TYPE, &pho, sizeof(MSG_p_init),ALL_NODES, NODE_PID);
  csend (P_LUM_TYPE, t_lum, n_lum*sizeof(Lumiere),ALL_NODES, NODE_PID);
  csend (P_PHO_TYPE, t_pho, n_pho*sizeof(Photo), ALL_NODES, NODE_PID);

  (void) free((char*) Tvoxlist);
  (void) free((char*) t_pol);
  (void) free((char*) seads);
}

do_comm ()
{
  int  nbpack, nx, ny, i;

  csend (GO_TYPE, NULL, 0, ALL_NODES, NODE_PID);
  nx = (int)ceil((double)xres/(double)NL);
  ny = (int)ceil((double)yres/(double)NL);
  nbpack = nx*ny;
  while (nbpack > 0) {
    /*
     * reception of nb pixel blocks.
     */
    crecv (COUL_TYPE, &msg_coul, sizeof(MSG_coul));
    for (i=0; i<msg_coul.nb; i++)
       ligne_out ((int)msg_coul.i[i],(int)msg_coul.j[i], msg_coul.buf[i]);
    nbpack -= msg_coul.nb;
  }
}
```



```
/*_____________________________________________________________________

  File          : host_data.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#include <stdio.h>
#include "ray_type.h"
#include "ray_comm.h"

int             yylinecount;
Vue             vue;
short           xres = 512;
short           yres = 512;

Lumiere         t_lum[MAXLUM];
Poly            *t_pol;
Photo           t_pho[MAXPHO];

int             seq = 1;

short           n_lum = 0;
short           n_pho = 0;
long            n_som = 0;
long            n_pol = 0;
long            ip = 0;
long            indice1 = 0;
long            indice2 = 0;

Color           fond   = {1.0,1.0,1.0};
Color           ambiant= {0.0,0.0,0.0};
Vue             vue;
Flt             rayeps = 1e-6 ;

short           profondeur = 5 ;
long            rayID = 1;

Voxlist         *Tvoxlist;
long            *seads;
long            size_seads;
short           nb_cell_x = SEADS;
short           nb_cell_y = SEADS;
short           nb_cell_z = SEADS;
short           coeff = COEFF;
long            ind_Tvox = 0;
long            poly_page;
long            maxvox = MAXVOX;
Flt             dx_vox, dy_vox, dz_vox;

Flt             x_min = MAXfloat;
Flt             y_min = MAXfloat;
Flt             z_min = MAXfloat;
Flt             x_max = MINfloat;
Flt             y_max = MINfloat;
Flt             z_max = MINfloat;

Flt             Poids = 0.0;

short           vite  = FAUX;
short           alias = FAUX;
short           outX  = FAUX;
```

```
short           outF  = FAUX;
short           zoom  = 1;

FILE            *fdres, *fdrvb;

short           maxrecu = 0;
```



```
/*_____________________________________________________________________

  File          : host_out.c
  Authors       : Thierry PRIOL & Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________

  Display the image during the computation on a X11 server                */

#include <stdio.h>
#include "ray_type.h"
#include "ray_comm.h"
#include "host_extern.h"

#ifdef  V_X11
#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/Xutil.h>

Display *screen;
Window window, wdstore;
XEvent event;
XImage *image;
Pixmap my_pixmap;
Colormap colours;
XColor colval;
int     xs, ys, DW, DH, Cx, Cy;
#endif

#define ABS(X)                  (((X) < 0) ? -(X) : (X))

unsigned char *data8;      /* 8 bits data image  */
unsigned char *data24;     /* 24 bits data image */

init_out ()
{
  int i;

  if (outF)     {
    data24 = (unsigned char*) malloc(xres*yres*3);
    if (data24 == NULL) {
      fprintf(stderr,"cannot alloc space for the bitmap 24 bits\n");
      outF = FAUX;
    } else {
      bzero ((char*) data24, xres*yres*3);
    }
  }
  if (!outX)    return;

#ifndef V_X11
  outX = FAUX;
#else
  xs = ys = 5;
  data8 = (unsigned char*) malloc(xres*yres);
  if (data8 == NULL) {
    fprintf(stderr,"cannot alloc space for the bitmap 8 bits\n");
    outX = FAUX;
    return;
  }
  bzero ((char*) data8, xres*yres);
  /*
   * Creating the window.
```

```
   */
  screen = XOpenDisplay (NULL);
  if (screen == NULL)   {
    fprintf(stderr,"cannot Open X Display \n");
    outX = FAUX;
    return;
  }
  DW = DisplayWidth(screen, DefaultScreen(screen));
  DH = DisplayHeight(screen, DefaultScreen(screen));
  Cx = (DW >> 2) + (DW >>1) + 5;
  Cy = (DH >> 1) + 5;
  window = XCreateSimpleWindow (screen, DefaultRootWindow(screen),
                                Cx-(xres>>1), Cy-(yres>>1),
                                xres, yres, 1, 0, 0);
  if (window == NULL)   {
    fprintf(stderr,"cannot Create X Window \n");
    outX = FAUX;
    return;
  }
  XChangeProperty(screen, window, XA_WM_NAME, XA_STRING, 8,
                  PropModeReplace, "IPSC/2 X11 Bitmap", 17);
  /*
   * 256 grey levels.
   */
  colours = XCreateColormap (screen, window,
                             DefaultVisual(screen, DefaultScreen(screen)),
                             AllocAll);
  colval.flags = DoRed | DoGreen | DoBlue;

  for (i=0; i<256; i++) {
    colval.pixel = i;
    colval.red = 256*i;
    colval.green = 256*i;
    colval.blue = 256*i;
    XStoreColor (screen, colours, &colval);
  }
  XMapWindow (screen, window);
  XSetWindowColormap (screen, window, colours);
  XInstallColormap (screen, colours);
  XSelectInput (screen, window, KeyPressMask|ExposureMask);
  image = XCreateImage (screen, DefaultVisual(screen, DefaultScreen(screen)),
                        8, ZPixmap, 0, data8, xres, yres, 8, 0);
  XPutImage (screen, window, DefaultGC(screen, DefaultScreen(screen)),
             image, 0, 0, 0, 0, xres, yres);
  XSync(screen,False);
#endif
}

ligne_out (li, lj, ligne)
     int li,lj;
     unsigned char *ligne;
{
  int    i,j,k,nbi,nbj;
  int    a8, a24, al;
  unsigned char r,v,b;

  if ((!outX)&&(!outF)) return;

  nbi = (li+NL>xres) ? xres - li : NL;
  nbj = (lj+NL>yres) ? yres - lj : NL;
  a8 = lj*xres + li;
  a24 = 3*a8;
  al = 0;

  for (j=0; j<nbj; j++, a8 += xres, a24 += 3*xres, al += 3*NL) {
    for (i=0, k=0; i<nbi; i++, k+= 3)   {
      r = ligne[al+k]; v = ligne[al+k+1]; b = ligne[al+k+2];
      /*
       * Grey = 29.9% Red + 58.7% Green + 11.4% Blue.
       */
      if (outX) {
        Flt fpix = (r*0.299 + v*0.587 + b*0.114);
        data8[a8 + i] = (fpix>254.0) ? 254: (unsigned char)fpix;
      }
      if (outF) {
        data24[a24+k] = r; data24[a24+k+1] = v; data24[a24+k+2] = b;
      }
    }
  }
#ifdef V_X11
  if (outX)    XPutImage(screen, window,
                         DefaultGC(screen, DefaultScreen(screen)),
                         image, li, lj, li, lj, NL, NL);
#endif
}

image_out ()
{
#ifdef V_X11
  if (outX)    XPutImage (screen, window,
                          DefaultGC(screen, DefaultScreen(screen)),
                          image, 0, 0, 0, 0, xres, yres);
#endif
  if (outF) {
    fprintf(stderr,"Saving the image file ...\n");
    fprintf(fdrvb,"%d %d\n", xres, yres);
    fwrite ((char*)data24, sizeof(char), xres*yres*3, fdrvb);
    fclose (fdrvb);
  }
#ifdef V_X11
  if (outX) {
    int done = FAUX;

    fprintf(stderr,"Press any Key in the window to stop\n");
    while(!done) {
      XNextEvent(screen,&event);
      switch (event.type) {
      case KeyPress :  done=VRAI;
        break;
      case Expose:
        if (event.xexpose.count == 0)
          XPutImage (screen, window, DefaultGC(screen, DefaultScreen(screen)),
                     image, 0, 0, 0, 0, xres, yres);
        break;
      case  MappingNotify:
        XRefreshKeyboardMapping(&event);
        break;
      }
    }
    XDestroyWindow(screen,window);
    XCloseDisplay(screen);
  }
#endif
}
```

```
/*_____________________________________________________________________

  File          : host_page.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#include <stdio.h>
#include <math.h>
#include "ray_type.h"
#include "ray_comm.h"
#include "ray_page.h"

#define LOG2(X)  (M_LOG2E*log((double)X))

static  int     objid = 0;
static  long    pagid = 0;
static  long    nodid = 0;
static  Objdesc objdesc;
static  long    maxpage;

OpenPage()
{
  maxpage = (PAGENUMB-1)*numnodes();
}

int CreateObj(size)
     int        size;
{
  double dv;
  long   lv;

  if (objid >= OBJNUMB) {
    fprintf(stderr,"cannot create a new object.\n");
    EXIT(-1);
  }
  dv = ceil(LOG2(size));
  lv = (long)dv;
  if (size != (1<<lv)) {
    fprintf(stderr,"size of object %d must be 2^n.\n",objid);
    EXIT(-1);
  }
  objdesc[objid].page = pagid;
  objdesc[objid].numb = (int)(PAGESIZE/size);
  objdesc[objid].size = size;
  objdesc[objid].dim  = lv;
  return (objdesc[objid++].numb);
}

SendPage(adr)
     char *adr;
{
  if (pagid >= maxpage) {
    fprintf(stderr,"\n please use more nodes.\n");
    EXIT(-1);
  }
  fprintf(stderr,"\tPage %d on Node %d\t\r",pagid,nodid);
  csend (PAGE_TYPE, adr, PAGESIZE, nodid, NODE_PID);
  pagid++;
  if (++nodid == numnodes())    nodid = 0;
}

SendObj(adr,size,nb)
     char       *adr;
     int     size;
     long    nb;
{
  long i;
  int no = CreateObj(size);
  int np = (int)ceil((double)nb/(double)no);
  for (i=0; i<np; i++, adr += PAGESIZE)
    SendPage(adr);
}

ClosePage()
{
  int i;

  for (i=0 ; i<objid; i++) {
    fprintf(stderr,"object %d:\t\t\t \n", i);
    fprintf(stderr,"\t page %d\n",objdesc[i].page);
    fprintf(stderr,"\t numb %d\n",objdesc[i].numb);
    fprintf(stderr,"\t size %d\n",objdesc[i].size);
    fprintf(stderr,"\t dim  %d\n",objdesc[i].dim);
  }
  fprintf(stderr,"%d pages used\n",pagid);
  csend (PAGE_TYPE, NULL, 0, ALL_NODES, NODE_PID);
  csend (OBJ_TYPE, objdesc, sizeof(Objdesc), ALL_NODES, NODE_PID);
}
```

```
/*_____________________________________________________________________

  File          : host_seads.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#include <stdio.h>
#include <math.h>
#include "ray_type.h"
#include "ray_comm.h"
#include "ray_page.h"
#include "host_extern.h"

#define CLAMP(A,MIN,MAX)  (A<MIN?MIN:(A>MAX?MAX:A))
#define ABS(X)            (((X) < 0) ? -(X) : (X))
#define X                 1
#define Y                 2
#define Z                 3
#define EPSIcmp       1e-05

char    *realloc();
short   imin, jmin, kmin;
short   imax, jmax, kmax;

static int depx, depy;

InitSeads ()
{
  double size_x, size_y, size_z, size_cell, vol_cell;
  long   nb_page_seads;
  /*
   * Parameters for the GRID volume.
   */
  size_x = x_max - x_min;
  size_y = y_max - y_min;
  size_z = z_max - z_min;
  vol_cell = (size_x * size_y * size_z) / (coeff * n_pol);
  size_cell = (Flt) exp(log((double)vol_cell)/3.0);
  nb_cell_x = (int) (size_x / size_cell);
  nb_cell_x = CLAMP (nb_cell_x, 1, SEADS);
  nb_cell_y = (int) (size_y / size_cell);
  nb_cell_y = CLAMP (nb_cell_y, 1, SEADS);
  nb_cell_z = (int) (size_z / size_cell);
  nb_cell_z = CLAMP (nb_cell_z, 1, SEADS);
  dx_vox = size_x / nb_cell_x;
  dy_vox = size_y / nb_cell_y;
  dz_vox = size_z / nb_cell_z;

  depx = nb_cell_y*nb_cell_z;
  depy = nb_cell_z;
  size_seads = nb_cell_x*nb_cell_y*nb_cell_z;
  nb_page_seads = (long)ceil((double)(size_seads*sizeof(long))/PAGESIZE);
  seads = (long *) malloc(nb_page_seads*PAGESIZE);
  if (seads == NULL) {
    fprintf(stderr,"can not alloc space for the SEADS.\n");
    exit (-1);
  }
  bzero ((char*)seads, size_seads*sizeof(long));
  fprintf(stderr,"\tGrid: %dX%dX%d\n",nb_cell_x,nb_cell_y,nb_cell_z);
}               /* Fin de init_seads */
```



```
RangerEltSeads ()
{
  int   i, j, k, indice;
  Flt   xmin,xmax,ymin,ymax,zmin,zmax;
  Point *point = t_pol[ip].point;
  long  pol = t_pol[ip].poly;

  xmin = ymin = zmin = MAXfloat;
  xmax = ymax = zmax = MINfloat;

  for (i=0; i<3; i++)   {
    if (point[i][0] < xmin)     xmin = point[i][0];
    if (point[i][0] > xmax)     xmax = point[i][0];
    if (point[i][1] < ymin)     ymin = point[i][1];
    if (point[i][1] > ymax)     ymax = point[i][1];
    if (point[i][2] < zmin)     zmin = point[i][2];
    if (point[i][2] > zmax)     zmax = point[i][2];
  }
  imin = (int)((xmin - x_min)/dx_vox - EPSIcmp);
  jmin = (int)((ymin - y_min)/dy_vox - EPSIcmp);
  kmin = (int)((zmin - z_min)/dz_vox - EPSIcmp);
  imax = (int)((xmax - x_min)/dx_vox + EPSIcmp);
  jmax = (int)((ymax - y_min)/dy_vox + EPSIcmp);
  kmax = (int)((zmax - z_min)/dz_vox + EPSIcmp);

  imin = CLAMP(imin, 0, nb_cell_x - 1);
  imax = CLAMP(imax, 0, nb_cell_x - 1);
  jmin = CLAMP(jmin, 0, nb_cell_y - 1);
  jmax = CLAMP(jmax, 0, nb_cell_y - 1);
  kmin = CLAMP(kmin, 0, nb_cell_z - 1);
  kmax = CLAMP(kmax, 0, nb_cell_z - 1);

  for (i = imin; i <= imax; i++) {
    for (j = jmin; j<= jmax; j++) {
      indice =  i*depx+j*depy + kmin;
      for (k = kmin; k<= kmax; k++, indice++) {
        /*
         * Storing the polygon in a voxel.
         */
        StoreInVoxel (pol,indice);
      }
    }
  }
}

StoreInVoxel (pol, i)
     long       pol;
     int        i;
{
  long  iv = seads[i];
  long  *vx;

  if (iv == 0)  {
    /*
     * Empty voxel.
     */
    if (ind_Tvox >= maxvox)     {
      maxvox += 16384;
      Tvoxlist = (Voxlist*)realloc(Tvoxlist,maxvox*sizeof(Voxlist));
      if (Tvoxlist == NULL) {
        fprintf (stderr,"filling SEADS polygons: no more free list\n");
        EXIT(-1);
      }
```

```
      fprintf(stderr,"\t\t\t\t realloc memory for voxels.(%d)\r",maxvox);
    }
    vx = Tvoxlist[ind_Tvox];
    seads[i] = ++ind_Tvox;
    vx[0] = pol;
    vx[1] = MAXlong;
    vx[NB_ELT_LIST - 1] = MAXlong;
  } else        {
    int l  = 1;
    /*
     * Go through a voxel chain.
     */
    while (Tvoxlist[iv-1][NB_ELT_LIST-1] != MAXlong) {
      iv = Tvoxlist[iv-1][NB_ELT_LIST-1];
    }
    vx = Tvoxlist[iv-1];
    /*
     * Searching a free place.
     */
    while (vx[l] != MAXlong)     l++;

    if (l == NB_ELT_LIST - 1)    {
      /*
       * Adding a list to a voxel.
       */
      if (ind_Tvox >= maxvox)    {
        maxvox += 16384;
        Tvoxlist = (Voxlist*)realloc(Tvoxlist,maxvox*sizeof(Voxlist));
        if (Tvoxlist == NULL) {
          fprintf (stderr,"filling SEADS polygons: no more free list\n");
          EXIT(-1);
        }
        fprintf(stderr,"\t\t\t\t realloc memory for voxels.(%d)\r",maxvox);
      }
      Tvoxlist[iv-1][NB_ELT_LIST-1] = ind_Tvox + 1;
      vx = Tvoxlist[ind_Tvox++];
      vx[NB_ELT_LIST-1] = MAXlong;
      l=0;
    }
    vx[l] = pol;
    if (++l != NB_ELT_LIST)      vx[l] = MAXlong;
  }

}
```



```
/*_____________________________________________________________________

  File          : host_extern.h
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

extern  short           xres, yres;

extern  Lumiere         t_lum[];
extern  Poly            *t_pol;
extern  Photo           t_pho[];

extern  Voxlist         *Tvoxlist;
extern  long            *seads;
extern  long            size_seads;
extern  short           nb_cell_x, nb_cell_y, nb_cell_z;
extern  short           coeff;
extern  long            ind_Tvox;
extern  long            poly_page;
extern  long            maxvox;
extern  Flt             dx_vox, dy_vox, dz_vox;

extern  short           n_lum;
extern  short           n_pho;
extern  long            n_som;
extern  long            n_pol;
extern  long            ip;
extern  long            indice1;
extern  long            indice2;

extern  Color           fond;
extern  Color           ambiant;
extern  Vue             vue;
extern  short           profondeur;
extern  long            rayID;

extern  Flt             x_min, x_max;
extern  Flt             y_min, y_max;
extern  Flt             z_min, z_max;

extern  Flt             Poids;

extern  FILE            *fdres, *fdrvb;

extern  short           maxrecu;
extern  int             seq;

extern  short           vite;
extern  short           alias;
extern  short           outX;
extern  short           outF;
extern  short           zoom;
extern  Flt             seuil;
```

```
# Makefile for MV_pRAY - Node program
# Generate code for an iPSC/2 with 4MBytes per nodes
# If your configuration is different, you should modify
# the value of PAGENUMB in the file ray_page.h.
#
PROG=node
CC=gcc
CFLAGS=-O

CSRC=node_page.c node_comm.c node_data.c node_main.c node_param.c \
     node_pixel.c node_plane.c node_photo.c node_pol.c node_seads.c \
     node_slabs.c
COBJ=node_page.o node_comm.o node_data.o node_main.o node_param.o \
     node_pixel.o node_plane.o node_photo.o node_pol.o node_seads.o \
     node_slabs.o

HDRS=ray_comm.h ray_def.h node_extern.h ray_type.h ray_vec.h
LIBS=-node -lm

$(PROG):$(COBJ) $(OOBJ)
        $(CC) $(CFLAGS) -o $(PROG) $(COBJ) $(LOBJ) $(LIBS)

clean:
        rm -f $(COBJ)
        rm -f core tags

lint:
        lint $(CSRC)
```



```
/*_____________________________________________________________________

  File          : node_main.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________
  If we make peacefule revolution impossible, we make violent revolution
  inevitable.                                        J.F. Kennedy        */

#include <stdio.h>
#include "../host/ray_type.h"
#include "../host/ray_comm.h"
#include "node_extern.h"

main ()
{
  int i;

  InitComm ();
  InitPol ();
  InitStart ();

  for (i=0; i<seq; i++) {
    lanceur ();
  }
}
```

```
/*_____________________________________________________________________

  File          : node_data.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#include <stdio.h>
#include "../host/ray_type.h"
#include "../host/ray_comm.h"

Rayon           *myray;

short           traiter = VRAI;

Flt             x_min = MAXfloat;
Flt             y_min = MAXfloat;
Flt             z_min = MAXfloat;
Flt             x_max = MINfloat;
Flt             y_max = MINfloat;
Flt             z_max = MINfloat;

Lumiere         t_lum[MAXLUM];
Photo           t_pho[MAXPHO];

long            n_pol = 0;
short           n_lum = 0;
short           n_pho = 0;

int             seq;

Color           fond   = {1.0,1.0,1.0};
Color           ambiant= {0.0,0.0,0.0};
Vue             vue;
Flt             rayeps = 1e-6 ;

short           profondeur = 5 ;
long            rayID = 0;

short           xres = 0;
short           yres = 0;
short           bx1,bx2,by1,by2;

short           vite  = FAUX;
short           alias = FAUX;
short           outX  = FAUX;
short           zoom  = 1;

long            *seads;
int             ind1, ind2;
short           nb_cell_x = SEADS;
short           nb_cell_y = SEADS;
short           nb_cell_z = SEADS;
Flt             dx_vox, dy_vox, dz_vox;

long            Me, num_nodes, node_dim;
```



```
/*_____________________________________________________________________

  File          : node_comm.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#include <stdio.h>
#include <math.h>
#include <cube.h>
#include "../host/ray_type.h"
#include "../host/ray_comm.h"
#include "../host/ray_page.h"
#include "node_extern.h"

static  long    Host;
static  short   nbuf;
static  long    nbpack = 0;
static  long    Pred, Succ;
static  int     envoi=FAUX;
static  long    msg,msg_bm,msg_wk,msgh,msgh2;
static  MSG_dem msg_dem_in,msg_dem_out,msg_pack_in;
static  MSG_p_init pho;
static  MSG_coul msg_coul;
static  MSG_pack pack,packext,packin;
static  short   packx_N, packy_N, x_N, y_N, dx, dy;
static  short   nb_N;
static  int     nbpixel;

extern  Flt  bitmap[NL*NL*3];
extern  Mem  mem;

void SendPack();
void SendElt();


GeoComm ()
{
  crecv (long_TYPE, &n_pol, sizeof(long));
  crecv (int_TYPE, &nb_cell_x, sizeof(int));
  crecv (int_TYPE, &nb_cell_y, sizeof(int));
  crecv (int_TYPE, &nb_cell_z, sizeof(int));
  crecv (Flt_TYPE, &dx_vox, sizeof(Flt));
  crecv (Flt_TYPE, &dy_vox, sizeof(Flt));
  crecv (Flt_TYPE, &dz_vox, sizeof(Flt));
  crecv (Flt_TYPE, &x_min, sizeof(Flt));
  crecv (Flt_TYPE, &x_max, sizeof(Flt));
  crecv (Flt_TYPE, &y_min, sizeof(Flt));
  crecv (Flt_TYPE, &y_max, sizeof(Flt));
  crecv (Flt_TYPE, &z_min, sizeof(Flt));
  crecv (Flt_TYPE, &z_max, sizeof(Flt));
  InitSeads();
}

PhotoComm ()
{
  crecv (P_INIT_TYPE, (char*)&pho, sizeof(MSG_p_init));
  crecv (P_LUM_TYPE, (char*)t_lum, pho.n_lum*sizeof(Lumiere));
  crecv (P_PHO_TYPE, (char*)t_pho, pho.n_pho*sizeof(Photo));
  xres = pho.xres;
  yres = pho.yres;
  vite = pho.vite;
```

```
  n_lum = pho.n_lum;
  profondeur = pho.profondeur;
  bcopy ((char*)pho.fond,(char*)fond, sizeof(Color));
  bcopy ((char*)pho.ambiant,(char*)ambiant, sizeof(Color));
  bcopy ((char*)&pho.vue,(char*)&vue, sizeof(Vue));
}

InitComm ()
{
  long i;
  int nl_Nx, nl_Ny, nx, ny, x, y;

  node_dim  = nodedim();
  num_nodes = numnodes();
  Me = mynode();
  Host = myhost();
  Succ = gray((ginv(Me)+1)%num_nodes);
  Pred = (Me == 0) ? num_nodes>>1 : gray((ginv(Me)-1)%num_nodes);
  packext.node  = Me;
  msg_dem_out.type = PACK_TYPE;
  msg_dem_out.n    = Me;
  x_N = y_N = 0;
  nbpixel = NL*NL*3;
  crecv (int_TYPE, &seq, sizeof(int));
  InitPage();
  GeoComm();
  PhotoComm();
  nx =  (int)ceil((double)xres/(double)NL);
  ny =  (int)ceil((double)yres/(double)NL);
  nl_Nx = (1<<(short)ceil((double)node_dim/2.0));
  nl_Ny = (1<<(short)floor((double)node_dim/2.0));
  x = (int)ceil((double)nx/(double)nl_Nx);
  y = (int)ceil((double)ny/(double)nl_Ny);
  dx = x*NL;
  dy = y*NL;
  packext.x = packx_N = (Me%nl_Nx)*dx;
  packext.y = packy_N = ((int)(Me/nl_Nx))*dy;
  pack.node = Me;
  nb_N = VRAI;
  nbuf = 0;
}

InitStart ()
{
  crecv (GO_TYPE, NULL, 0);
  msg_wk = irecv (PACK_TYPE, &packin, sizeof(MSG_pack));
  msg = isend (PACK_TYPE, &packext, sizeof(MSG_pack), Pred, NODE_PID);
  while (!msgdone(msg));
  led(0);
  hrecv (DEM_TYPE, &msg_dem_in, sizeof(MSG_dem), SendElt);
  hrecv (DEM_PACK, &msg_pack_in, sizeof(MSG_dem), SendPack);
}

void SendElt (t,c,n,p)
     long t,c,n,p;
{
  msgh = isend (PAGE_TYPE, mem[msg_dem_in.n], PAGESIZE, n, p);
  while (!msgdone(msgh));
  hrecv (DEM_TYPE, &msg_dem_in, sizeof(MSG_dem), SendElt);
}

void SendPack (t,c,n,p)
     long t,c,n,p;
{
  if (msg_pack_in.n == Me) {
    packext.x = -1;
    msgh2 = isend (PACK_TYPE, &packext, sizeof(MSG_pack), msg_pack_in.n, p);
  } else if (GetPack(&packext)) {
    msgh2 = isend (PACK_TYPE, &packext, sizeof(MSG_pack), msg_pack_in.n, p);
  } else {
    msgh2 = isend (DEM_PACK, &msg_pack_in, sizeof(MSG_dem), Succ, p);
  }
  while (!msgdone(msgh2));
  hrecv (DEM_PACK, &msg_pack_in, sizeof(MSG_dem), SendPack);
}

int GetPack(pk)
     MSG_pack *pk;
{
  long oldmask;

  if (nb_N == FAUX) return (FAUX);
  oldmask = masktrap (DEM_PACK);
  if (nb_N == VRAI) {
    if (((x_N += NL)>=dx)||(packx_N+x_N >= xres)) {
      if (((y_N += NL)>=dy)||(packy_N+y_N >= yres)) {
        y_N = 0;
        nb_N = FAUX;
      }
      x_N = 0;
    }
    if (nb_N == VRAI) {
      pk->x = packx_N + x_N;
      pk->y = packy_N + y_N;
      pk->node = Me;
    }
  }
  masktrap (oldmask);
  return ((int)nb_N);
}

GetWork()
{
  if (!GetPack(&pack)) {
    while (!msgdone(msg_wk));
    if (packin.x == -1) EndWork ();
    bx1 = packin.x; bx2 = bx1 + NL;
    if (bx2 > xres) bx2 = xres;
    by1 = packin.y; by2 = by1 + NL;
    if (by2 > yres) by2 = yres;
    msg = isend (DEM_PACK, &msg_dem_out, sizeof(MSG_dem), packin.node, NODE_PID);
    msg_wk = irecv (PACK_TYPE, &packin, sizeof(MSG_pack));
    while (!msgdone(msg));
  } else {
    bx1 = pack.x; bx2 = bx1 + NL;
    if (bx2 > xres) bx2 = xres;
    by1 = pack.y; by2 = by1 + NL;
    if (by2 > yres) by2 = yres;
  }
}

EndWork ()
{
  if (nbuf > 0) {
    msg_coul.nb   = nbuf;
    msg_bm = isend (COUL_TYPE,&msg_coul,sizeof(MSG_coul),Host,HOST_PID);
```

```
    nbuf   = 0;
    while (!msgdone(msg_bm));
  }
  while (VRAI);
}

GiveWork ()
{
  int   n, i;
  Flt   val;
  unsigned char *buf = msg_coul.buf[nbuf];

  if (envoi) {while (!msgdone(msg_bm)); envoi = FAUX;}
  msg_coul.i[nbuf] = bx1;
  msg_coul.j[nbuf] = by1;
  for (n=0; n<nbpixel; n++) {
    val = bitmap[n];
#ifdef USE_GAM
    /*
     * val = (val*gain)^(1/gamma)
     */
    val = exp(log(val*1.3)/2.3);
#endif
    buf[n] = (val > 1.0) ? 255 : (unsigned char)(val*255.0);
  }
  nbuf++;
  if (nbuf == NB) {
    msg_coul.nb   = nbuf;
    msg_bm = isend (COUL_TYPE,&msg_coul,sizeof(MSG_coul),Host,HOST_PID);
     envoi  = VRAI;
    nbuf   = 0;
  }
  nbpack++;
}
```



```
/*_____________________________________________________________________

  File          : node_page.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#include <stdio.h>
#include <math.h>
#include "../host/ray_type.h"
#include "../host/ray_comm.h"
#include "../host/ray_page.h"
#include "node_extern.h"

#define MODULO_2n(A,N)   (A & (N-1))

#if (WAY == 1)
/*
 * Direct mapping policy.
 */
static  Elt     tabcach[ENTRY];
#else
/*
 * Set associative policy.
 */
static  Elt     tabcach[ENTRY][WAY];
#endif
static  long    pagid = 0;
static  long    mypage;
static  int     fault;
        Objdesc objdesc;

static MSG_dem msg_dem;
static long msg_in,msg_out;

Mem mem;        /* Memory place for data */
long entry;

#if (WAY > 1)
static long date;
int compar (a,b)
     Elt *a, *b;
{
  return (a->date > b->date);
}
#endif

InitPage ()
{
  long i,j;

  do {
     crecv (PAGE_TYPE, mem[pagid++], PAGESIZE);
  } while (infocount());
  crecv (OBJ_TYPE, objdesc, sizeof(Objdesc));

  entry = (int)((PAGENUMB - pagid)/WAY);
  if (Me == 0) fprintf(stderr,"cache[%d] %d-WAY\n",entry,WAY);

  mypage = pagid;
#if (WAY == 1)
  for(i=0; i<entry; i++) {
```

```
    tabcach[i].index = MAXlong;
    tabcach[i].entree = pagid++;
  }
#else
  for(i=0; i<entry; i++) {
    for(j=0; j<WAY; j++) {
      tabcach[i][j].date = 0;
      tabcach[i][j].index = MAXlong;
      tabcach[i][j].entree = pagid++;
    }
  }
  date = 0;
#endif
  msg_dem.type = PAGE_TYPE;
}

char *GetAdObj (id, num)
     int  id;
     long num;
{
  long  ip, in, dn, dp;
  int   ent,i,numb;
  char  *ad;

  numb = objdesc[id].numb;
  ip = objdesc[id].page + (long)(num/numb);
  in = MODULO_2n(ip,num_nodes);
  dn = (long)(ip>>node_dim);
  dp = num%numb;

  if (in==Me)   {
    /*
     * local data.
     */
    ad = mem[dn];
  } else {
    ent = (int)(ip%entry);
    fault = VRAI;
#if (WAY == 1)
  if (ip == tabcach[ent].index) { /* donnee dans le cache */
    ad = mem[tabcach[ent].entree];
    fault = FAUX;
  }
#else
    date++;
    for (i=0; i<WAY; i++) {
      if (ip == tabcach[ent][i].index) {
        /*
         * data in the cache.
         */
        tabcach[ent][i].date = date;
        ad = mem[tabcach[ent][i].entree];
        fault = FAUX;
        break;
      }
    }
#endif
    if (fault) {
      /*
       * page fault.
       */
      msg_dem.n = dn;
      msg_out = isend (DEM_TYPE, &msg_dem, sizeof(MSG_dem), in, NODE_PID);
#if (WAY == 1)
      ad = mem[tabcach[ent].entree];
      tabcach[ent].index = ip;
#else
      ad = mem[tabcach[ent][0].entree];
      tabcach[ent][0].date = date;
      tabcach[ent][0].index = ip;
#endif
      msg_in = irecv (PAGE_TYPE, ad, PAGESIZE);
#if (WAY > 1)
      qsort ((char*)tabcach[ent], WAY, sizeof(Elt), compar);
#endif
      led(1);
      while (!msgdone(msg_out));
      while (!msgdone(msg_in));
      led(0);
      if (id == 0) {
        Poly *pp = (Poly *)ad;
        pp->rayID = 0;
      }
    }
  }
  if (id == 2) {
    bcopy ( ad, (char *)seads, PAGESIZE);
    ind1 = num - dp;
    ind2 = ind1 + numb - 1;
  }
  ad += dp<<objdesc[id].dim;
  return (ad);
}
```

```
/*_____________________________________________________________________

  File          : node_param.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#define EPSIcmp         1e-05
#define SUPEG(A,B)      ((A >= (B - EPSIcmp)) ? 1 : 0)
#define INFEG(A,B)      ((A <= (B + EPSIcmp)) ? 1 : 0)

#include        <stdio.h>
#include        "../host/ray_type.h"
#include        "../host/ray_comm.h"
#include        "node_extern.h"

static short f_plan,i1,i2;
static Point *s, Q;
static Flt s0u, s0v;
static int bool_inter;
static Flt alpha,beta;
static Flt u0,u1,u2,v0,v1,v2;

InterRayParam (fp, t)
     /*
      * Parametric resolution for ray/triangle intersection.
      */
     Poly *fp;
     Flt t;
{
  f_plan= fp->f_plan ;
  i1    = fp->i1;
  i2    = fp->i2;
  s     = fp->point;
  s0u   = s[0][i1];
  s0v   = s[0][i2];

  Q[0] = myray->ori[0] + myray->dir[0]*t;
  Q[1] = myray->ori[1] + myray->dir[1]*t;
  Q[2] = myray->ori[2] + myray->dir[2]*t;

  u0 = Q[i1] - s0u;
  v0 = Q[i2] - s0v;

  bool_inter = FAUX;
  u1 = s[1][i1] - s0u;
  v1 = s[1][i2] - s0v;
  u2 = s[2][i1] - s0u;
  v2 = s[2][i2] - s0v;

  if (u1 == 0)  {
    beta = u0/u2;
    if (SUPEG(beta,0)&&INFEG(beta,1))   {
      alpha = (v0 - beta*v2)/v1;
      bool_inter = (SUPEG(alpha,0)&&INFEG(alpha+beta,1));
    }
  }
  else          {
    beta = (v0*u1 - u0*v1)/(v2*u1 - u2*v1);
    if (SUPEG(beta,0)&&INFEG(beta,1))   {
      alpha = (u0 - beta*u2)/u1;
      bool_inter = (SUPEG(alpha,0)&&INFEG(alpha+beta,1));
    }
  }

  if (bool_inter)       {
    /*
     * The intersection point is inside the polygon.
     */
    bcopy ((char *)Q, (char *)myray->point, sizeof(Point));

    if (!f_plan)        {
      /*
       * Interpolation of the normal vector.
       */
      Flt               *n0 = fp->normal[0];
      Flt               *n1 = fp->normal[1];
      Flt               *n2 = fp->normal[2];
      Flt                gamm = 1 - (alpha+beta);

      myray->normal[0] = gamm*n0[0] + alpha*n1[0] + beta*n2[0];
      myray->normal[1] = gamm*n0[1] + alpha*n1[1] + beta*n2[1];
      myray->normal[2] = gamm*n0[2] + alpha*n1[2] + beta*n2[2];
    } else              {
      bcopy ((char *)fp->plan, (char *)myray->normal, sizeof(Vec));
    }
  }

  return (bool_inter);
}
```

```
/*_____________________________________________________________________

  File          : node_photo.c
  Authors       : T.Priol & B.Arnaldi & D. Badouel
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________
  Mirrors should reflect a little before throwing back images.
                                                    Jean Cocteau        */

#include <stdio.h>
#include <math.h>
#include "../host/ray_type.h"
#include "../host/ray_vec.h"
#include "../host/ray_comm.h"
#include "node_extern.h"

#define EPSIlum                 EPSILON
#define EVALCONTINUE(coef)      ((coef[0] > EPSIlum)&&\
                                 (coef[1] > EPSIlum)&&\
                                 (coef[2] > EPSIlum))
#define SQR(x)                  ((x) * (x))
#define ABS(X)                  (((X) < 0) ? -(X) : (X))

RayPhoto (ray)
     Rayon              *ray;
{
  Color         color;

  intersect(ray);
  if ((ray->type == PRIM)||(ray->type == SECON))        {
    /*
     * Primary or secondary ray (no light ray).
     */
    Vec         bissectrice, coeff;
    Flt         sca, diffuse, speculaire, angle_light;
    short       i;
    int         gointer;
    Rayon       r2;

    if(ray->t != MAXfloat)      {
      Photo     * photo;
      /*
       * Ambiant contribution to the pixel.
       */
      photo = &(t_pho[ray->photo]);
      COMB_VEC (ray->coeff, ambiant, color);
      ALPHA_VEC ( photo->kd, color, color);
      COMB_VEC (photo->color, color, color);
      PixelIma (ray->ipix, ray->jpix, color);
      /*
       * Origin point for secondary and light rays.
       */
      bcopy((char*)ray->point, (char*)r2.ori, sizeof(Point));
      r2.ipix = ray->ipix;
      r2.jpix = ray->jpix;
      r2.poly = ray->poly;
      /*
       * Computing light sources contribution.
       */
      r2.type = LUM;
      if (PROD_SCAL(ray->dir,ray->normal) < 0.0) INV_VEC(ray->normal);
      for (i = 0; i < n_lum; i++) {
        r2.dir[0] = (t_lum[i].pos[0] - r2.ori[0]);
        r2.dir[1] = (t_lum[i].pos[1] - r2.ori[1]);
        r2.dir[2] = (t_lum[i].pos[2] - r2.ori[2]);

        r2.maxlum = TAILLE_VEC (r2.dir);
        if (r2.maxlum > EPSILON) {
          r2.dir[0] /= r2.maxlum;
          r2.dir[1] /= r2.maxlum;
          r2.dir[2] /= r2.maxlum;
        }
        switch (t_lum[i].type)  {
        case SPOT       :
          angle_light= PROD_SCAL(r2.dir,t_lum[i].dir);
          if (angle_light > 0.0)        gointer = VRAI;
          else                          gointer = FAUX;
          break;
        case LUMI       :
          gointer = VRAI;
          angle_light = 1.0;
          break;
        }
        if (gointer)    {
          bissectrice[0] = (r2.dir[0] - ray->dir[0]) / 2.0;
          bissectrice[1] = (r2.dir[1] - ray->dir[1]) / 2.0;
          bissectrice[2] = (r2.dir[2] - ray->dir[2]) / 2.0;
          NORM_VEC (bissectrice);
          /*
           * Computing the diffuse part.
           */
          diffuse = ABS (PROD_SCAL (ray->normal, r2.dir)) * photo->kd;
          /*
           * Computing the reflection part.
           */
          sca = ABS (PROD_SCAL (ray->normal, bissectrice));
          if (sca > EPSILON)
            speculaire = pow((double)sca,(double)photo->phong) * photo->ks;
          else  speculaire = 0.0;
          /*
           * Casting a light ray.
           */
          coeff[0] = diffuse*photo->color[0] + speculaire;
          coeff[1] = diffuse*photo->color[1] + speculaire;
          coeff[2] = diffuse*photo->color[2] + speculaire;
          ALPHA_VEC (t_lum[i].coeff, coeff, r2.coeff);
          COMB_VEC  (r2.coeff, ray->coeff, r2.coeff);
          if (vite) {
            PixelIma (r2.ipix, r2.jpix, r2.coeff);
          } else    {
            r2.rayID  = rayID++;
            r2.ombre  = 1.0;
            RayPhoto (&r2);
          }
        }
      }

      if ((ray->niv > 1) && EVALCONTINUE(ray->coeff))   {
        /*
         * The light contribution is suffisant to spawn
         * secondary rays.
         */
        r2.type = SECON;
        r2.niv  = ray->niv - 1;
        /*
         * Reflection contribution.
```

```
         */
        if (photo->ks != 0.0) {
          calcdirspec (ray->normal, ray, r2.dir);
          ALPHA_VEC (photo->ks, ray->coeff, r2.coeff);
          r2.rayID = rayID++;
          RayPhoto (&r2);
        }
        /*
         * Refraction contribution.
         */
        if ((photo->ior != 0.0) && (photo->kt != 0.0)) {
          if (calcdirtrans (ray->normal,ray,r2.dir,1.0,photo->ior)) {
            ALPHA_VEC (photo->kt, ray->coeff, r2.coeff);
            r2.rayID = rayID++;
            RayPhoto (&r2);
          }
        }
      }
    } else      {
      /*
       * No intersection.
       */
      COMB_VEC (ray->coeff, fond, color);
      PixelIma (ray->ipix, ray->jpix, color);
    }
  } else        {
    /*
     * case of a light ray.
     */
    ALPHA_VEC (ray->ombre, ray->coeff, ray->coeff);
    PixelIma (ray->ipix, ray->jpix, ray->coeff);
  }
}


calcdirspec (n,r,v)
     Vec n;
     Rayon *r;
     Vec v;
{
  Flt   x;

  x = PROD_SCAL (n, r->dir);
  v[0] = r->dir[0] - 2.0 * x * n[0];
  v[1] = r->dir[1] - 2.0 * x * n[1];
  v[2] = r->dir[2] - 2.0 * x * n[2];
  NORM_VEC (v);
}

calcdirtrans (n,r,v,n1,n2)
     Vec n;
     Rayon *r;
     Vec v;
     Flt n1, n2;
{
  int   valide;
  Vec   rvectn;
  Flt   refract, rscaln, normrvectn, coef;
  short   orientn;

  rscaln = PROD_SCAL (r -> dir, n);
  orientn = (rscaln > 0.0) ? -1 : 1;
  /*
   * To compute, we give an orientattion for n.
   */
  refract = n1 / n2;
  PROD_VEC (r -> dir, n, rvectn);
  normrvectn = sqrt((double)PROD_SCAL (rvectn, rvectn));
  if (refract * normrvectn > 1.0)
    /*
     * Total reflection.
     */
    valide = FAUX;
  else {
    /*
     * Computing the refraction direction.
     */
    coef = (Flt ) (orientn) * (refract * ABS (rscaln) -
                               sqrt((double)1.0 - SQR (refract * normrvectn)));
    v[0] = refract * r->dir[0] + coef * n[0];
    v[1] = refract * r->dir[1] + coef * n[1];
    v[2] = refract * r->dir[2] + coef * n[2];
    valide = VRAI;
    NORM_VEC (v);
  }
  return (valide);
}
```

```
/*_____________________________________________________________________

  File          : node_pixel.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#include <stdio.h>
#include <math.h>
#include "../host/ray_type.h"
#include "../host/ray_comm.h"
#include "node_extern.h"
#include "../host/ray_vec.h"

extern int ligne;

Flt bitmap[NL*NL*3];

RazIma ()
{
  bzero((char*)bitmap,NL*NL*3*sizeof(Flt));
}

PixelIma (i, j, coul)
     short      i,j;
     Color      coul;
{
  int   a = 3*((j-by1)*NL + (i-bx1));

  bitmap[a    ] += coul[0];
  bitmap[a + 1] += coul[1];
  bitmap[a + 2] += coul[2];
}

Rayon   raypix;
static  Flt     incr;
static  Vec     vec_vue, vec_gauche, dir;
/*
 * Casting the primary rays.
 */
lanceur ()
{
  short   x,y;
  /*
   * Computing the view parameters.
   */
  SUB_VEC  (vue.at, vue.from, vec_vue);
  NORM_VEC (vec_vue);
  PROD_VEC (vue.up, vec_vue, vec_gauche);
  NORM_VEC (vec_gauche);
  PROD_VEC (vec_vue, vec_gauche, vue.up);
  NORM_VEC (vue.up);

  rayID = 1;
  bcopy ((char *) vue.from, (char *) raypix.ori, sizeof(Point));
  incr = ((Flt) tan(vue.angle));

  raypix.type  = PRIM;
  raypix.niv   = profondeur;
  raypix.coeff[0] = 1.0;
  raypix.coeff[1] = 1.0;
  raypix.coeff[2] = 1.0;

  while (VRAI) {
    RazIma ();
    GetWork();
    for (y=by1; y<by2; y++) {
      raypix.jpix  = y;
      for (x=bx1; x<bx2; x++) {
        raypix.rayID = rayID++;
        raypix.ipix = x;
        ALPHA_COMB_VEC (-incr * (2.0*(y-1)/(Flt)(yres-1) - 1.0), vue.up,
                        incr * (2.0*(x-1)/(Flt)(xres-1) - 1.0), vec_gauche,
                        dir);
        ADD_VEC  (dir, vec_vue, raypix.dir);
        NORM_VEC (raypix.dir);
        RayPhoto(&raypix);
      }
    }
    GiveWork();
  }
}
```

```
/*_____________________________________________________________________

  File          : node_plane.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#include <stdio.h>
#include <math.h>
#include "../host/ray_type.h"
#include "../host/ray_vec.h"
#include "../host/ray_comm.h"
#include "node_extern.h"

#define EPSIgeo                 EPSILON
#define EPSIlum                 EPSILON
#define EPSIcmp                 1e-05
#define EVALCONTINUE(coef)      ((coef[0] > EPSIlum)&&\
                                 (coef[1] > EPSIlum)&&\
                                 (coef[2] > EPSIlum))
#define SUP(A,B)                (((A - EPSIcmp) > B)  ? 1 : 0)
#define INF(A,B)                (((A + EPSIcmp) < B)  ? 1 : 0)

char *GetAdObj();

static Poly  *facet;
static Flt  near, far;
static Flt  t,u,v,w,d,x_0,y_0,z_0,dx,dy,dz,val;
static int  inter,final;

inter_ray_facet (poly)
     /*
      * Final intersection --> return (VRAI)
      * Else               --> return (FAUX)
      */
     long poly;
{
  facet = (Poly *)GetAdObj(0,poly);

  if ((facet->rayID == myray->rayID)||(myray->poly == poly)) {
    /*
     * Intersection already computed.
     */
    return (FAUX);
  } else {
    facet->rayID = myray->rayID;
  }

  far = (myray->type == LUM) ? MAXfloat : myray->t;
  near = 0.0;
  if (! InterPolySlabs (facet, &near, &far))    {
    return (FAUX);
  }
  return (InterRayPlane (poly));
}

InterRayPlane (poly)
     /*
      * Final intersection --> return (VRAI)
      * Else               --> return (FAUX)
      */
     long poly;
{
  u = facet->plan[0];v = facet->plan[1];w = facet->plan[2];d = facet->plan[3];
  x_0 = myray->ori[0];y_0 = myray->ori[1];z_0 = myray->ori[2];
  dx  = myray->dir[0];dy  = myray->dir[1];dz  = myray->dir[2];

  if( (val = u*dx + v*dy + w*dz) == 0 ) {
    /*
     * The ray and the polygon are \\.
     */
    return (FAUX)       ;
  }
  /*
   * Computing the t value at the impact point.
   */
  if( (t = -(x_0*u + y_0*v + z_0*w + d)/val) < EPSIgeo )        {
    /*
     * The impact is not in the ray direction: reject.
     */
    return (FAUX)       ;
  }
  if (INF(t,near) || SUP(t,far))        {
    /*
     * The point is out the slabs volume: reject.
     */
    return (FAUX)       ;
  }
  if (( myray->type == LUM )&&( t >= myray->maxlum ))   {
    /*
     * Intersection behind the light source.
     */
    return (FAUX)       ;
  }
  inter = InterRayParam ( facet, t);
  final = FAUX;
  if (inter)    {
    myray->photo = facet->photo;
    myray->t = t;
    myray->poly = poly;
    if (myray->type == LUM)     {
      Flt       kt = t_pho[myray->photo].kt;

      if ((myray->ombre *= kt) < EPSIlum)       myray->ombre = 0.0;
      final = (myray->ombre == 0.0);
    }
  }
  return (final);
}
```

```
/*_____________________________________________________________________

  File          : node_pol.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#include <stdio.h>
#include <math.h>
#include "../host/ray_vec.h"
#include "../host/ray_type.h"
#include "../host/ray_comm.h"
#include "../host/ray_page.h"
#include "node_extern.h"

#define ABS(X)          (((X) < 0) ? -(X) : (X))

extern Objdesc objdesc;

char *GetAdObj();

InitPol ()
{
  long  i,j;
  Poly *pp;
  long incr = objdesc[0].numb*num_nodes;

  for(i=Me*objdesc[0].numb; i<n_pol; i+=incr) {
    pp = (Poly *) GetAdObj(0,i);
    for (j=0; j<objdesc[0].numb; j++,pp++) {
      InitSlabs (pp);
      EquationPlan (pp);
      IndicePlan (pp);
    }
  }
}

EquationPlan (pp)
     Poly *pp;
{
  Vec   u, v, w;
  /*
   * Computing the normal vector for the plane.
   */
  VECTEUR (pp->point[0],pp->point[1],u);
  VECTEUR (pp->point[0],pp->point[2],v);
  PROD_VEC(u,v,w);
  NORM_VEC(w);

  pp->plan[0] = w[0];
  pp->plan[1] = w[1];
  pp->plan[2] = w[2];
  pp->plan[3] = -(pp->point[1][0]*w[0] + pp->point[1][1]*w[1]
                  + pp->point[1][2]*w[2]);
}

IndicePlan (pp)
     Poly *pp;
{
  /*
   * i0 represent the axe of 'biggest' projection.
   *       |a| > {|b|,|c|} -> axe X -> i0 = 0
   *       |b| > {|a|,|c|}  -> axe Y -> i0 = 1
   *       |c| > {|a|,|b|}  -> axe Z -> i0 = 2
   * i1 and i2 represent the two other direction.
   */
  if ( ABS(pp->plan[0]) > ABS(pp->plan[1]) )
    if ( ABS(pp->plan[0]) > ABS(pp->plan[2]) )  {
      pp->i1 = 1; pp->i2 = 2;
    } else  {
      pp->i1 = 0; pp->i2 = 1;
    }
  else    if ( ABS(pp->plan[1]) > ABS(pp->plan[2]) )  {
    pp->i1 = 2; pp->i2 = 0;
  } else  {
    pp->i1 = 0; pp->i2 = 1;
  }
}
```

```
/*_____________________________________________________________________

  File          : node_seads.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#include <stdio.h>
#include <math.h>
#include "../host/ray_page.h"
#include "../host/ray_type.h"
#include "node_extern.h"

#define EPSIcmp      1e-05
#define CLAMP(A,MIN,MAX)  (A<MIN?MIN:(A>MAX?MAX:A))
#define ABS(X)            (((X) < 0) ? -(X) : (X))
#define X                 1
#define Y                 2
#define Z                 3

char *GetAdObj();

static long voxel;
static int stepx,stepy,stepz;       /* Increments wrt ray direction. */
static int stepix,stepiy,stepiz;
static int indice;
static int x,y,z;                   /* Indices of the current voxel. */
static Flt t_in,t_out;              /* t input(output) of the voxel. */
static Flt tx,ty,tz;                /* t value wrt the axes.         */
static Flt deltax,deltay,deltaz;    /* increases between 2 voxels.   */
static Voxlist advox;
static int depx, depy;

InitSeads ()
{
  depx = nb_cell_y*nb_cell_z;
  depy = nb_cell_z;
  ind1 = ind2 = -1;
  seads = (long *) malloc(PAGESIZE);
  if (seads == NULL) {
    fprintf(stderr,"node %d:can not alloc a page for the SEADS.\n", Me);
    exit (-1);
  }
}

intersect (ray)
     Rayon *ray;
{
  myray = ray;
  myray->t = MAXfloat;
  myray->poly = -1;
  InterRaySlabs();
  InterRaySeads();
}

InterRaySeads ()
{
  /*
   *    1) Initialisations
   */
  {
    Flt x_0 = myray->ori[0], y_0 = myray->ori[1], z_0 = myray->ori[2];
    Flt dx = myray->dir[0], dy = myray->dir[1], dz = myray->dir[2];
    Flt a;
    short fin;

    stepx = (dx > 0) ? (1) : (-1);
    stepy = (dy > 0) ? (1) : (-1);
    stepz = (dz > 0) ? (1) : (-1);

    stepix = depx*stepx;
    stepiy = depy*stepy;
    stepiz = stepz;

    deltax = (dx == 0.) ? (MAXfloat) : (dx_vox/ABS(dx));
    deltay = (dy == 0.) ? (MAXfloat) : (dy_vox/ABS(dy));
    deltaz = (dz == 0.) ? (MAXfloat) : (dz_vox/ABS(dz));

    t_out = 0.0;
    do {
      fin = VRAI;
      if (x_0 < x_min)  {
        if (stepx <= 0) return;
        a = (x_min - x_0)/dx;
        x_0 = x_min; y_0 += a*dy; z_0 += a*dz;
        t_out += a;
      } else    {
        if (x_0 > x_max)        {
          if (stepx >= 0)       return;
          a   = (x_max - x_0)/dx;
          x_0  = x_max; y_0 += a*dy; z_0 += a*dz;
          t_out += a;
        }
      }

      if (y_0 < y_min)  {
        if (stepy <= 0) return; fin = FAUX;
        a   = (y_min - y_0)/dy;
        y_0  = y_min; x_0 += a*dx; z_0 += a*dz;
        t_out += a;
      } else    {
        if (y_0 > y_max)        {
          if (stepy >= 0)       return; fin = FAUX;
          a   = (y_max - y_0)/dy;
          y_0  = y_max; x_0 += a*dx; z_0 += a*dz;
          t_out += a;
        }
      }

      if (z_0 < z_min)  {
        if (stepz <= 0) return; fin = FAUX;
        a   = (z_min - z_0)/dz;
        z_0  = z_min; y_0 += a*dy; x_0 += a*dx;
        t_out += a;
      } else    {
        if (z_0 > z_max)        {
          if (stepz >= 0)       return; fin = FAUX;
          a   = (z_max - z_0)/dz;
          z_0  = z_max; y_0 += a*dy; x_0 += a*dx;
          t_out += a;
        }
      }
    } while (!fin);
    /*
     * Computing tx, ty, tz.
```

```
     */
    a = (x_0 - x_min)/dx_vox;
    x = (int)a; x = CLAMP(x,0,nb_cell_x-1);
    if (stepx > 0) tx = ((x+1) - a)*deltax;
    else           tx = (a - x)*deltax;

    a = (y_0 - y_min)/dy_vox;
    y = (int)a; y = CLAMP(y,0,nb_cell_y-1);
    if (stepy > 0) ty = ((y+1) - a)*deltay;
    else           ty = (a - y)*deltay;

    a = (z_0 - z_min)/dz_vox;
    z = (int)a; z = CLAMP(z,0,nb_cell_z-1);
    if (stepz > 0)  tz = ((z+1) - a)*deltaz;
    else            tz = (a - z)*deltaz;

    tx += t_out; ty += t_out; tz += t_out;
    indice = x*depx + y*depy +z;
  }
  {
    unsigned short      dir;
    /*
     *  2) Incremental ruuning through the grid.
     */
    while (VRAI) {
      /*
       * Searching the next stepping direction.
       */
      t_in = t_out;
      if (tx < ty) {
        if (tx < tz) { t_out = tx; dir  = X; }
        else         { t_out = tz; dir  = Z; }
      } else {
        if (ty < tz) { t_out = ty; dir  = Y; }
        else         { t_out = tz; dir  = Z; }
      }
      if ((indice < ind1) || (indice > ind2)) {
        /*
         * Steping in an other page.
         */
        (void) GetAdObj (2,indice);
      }
      voxel = seads[indice - ind1];
      if (voxel != 0) {
        /*
         * Intersecting a non-empy voxel.
         */
        if (IntersectPolys (voxel)) return;
      }
      /*
       * Steping in the next voxel.
       */
      switch (dir)      {
      case      X  :
        x += stepx;
        if ((x < 0)||(x >= nb_cell_x))
          /*
           * Going out the grid.
           */
          return;
        indice += stepix;
        tx += deltax;
        break;

      case      Y  :
        y += stepy;
        if ((y < 0)||(y >= nb_cell_y))
          /*
           * Going out the grid.
           */
          return;
        indice += stepiy;
        ty += deltay;
        break;

      case      Z  :
        z += stepz;
        if ((z < 0)||(z >= nb_cell_z))
          /*
           * Going out the grid.
           */
          return;
        indice += stepiz;
        tz += deltaz;
        break;
      }
    }
  }
}

IntersectPolys (iv)
     /*
      * Compute the intersection between a ray and a
      * list of polygon.
      */
     long               iv;
{
  int nb_elt, fin;

  bcopy ( GetAdObj(1,iv-1),(char*)advox, sizeof(Voxlist));
  nb_elt  = 0;
  fin   = FAUX;
  while (VRAI)  {
    if ( nb_elt == NB_ELT_LIST - 1)     {
      /*
       * Go through a voxel chain.
       */
      iv = advox[NB_ELT_LIST-1];
      if ((iv != MAXlong)&&(iv > 0)) {
        bcopy ( GetAdObj(1,iv-1),(char*)advox, sizeof(Voxlist));
        nb_elt = 0;
      } else {
        fin = VRAI;
      }
    } else {
      fin = ( advox[nb_elt] == MAXlong );
    }

    if (fin)
      break;
    if ( inter_ray_facet(advox[nb_elt]))
      return (VRAI);
    nb_elt++;
  }
  /*
   * Is the intersection before the exit of the voxel ?
```

```
   * If true, this is the nearest intersection to the ray origin.
   */
  return (( myray->type != LUM)&&( myray->t <= t_out ));

}
```



```
/*_____________________________________________________________________

  File          : node_slabs.c
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
_____________________________________________________________________*/
/*
 * For more details see "Ray Tracing Complex Scenes" T.L. Kay&J.T. Kajiya.
 */
#include        <stdio.h>
#include        <math.h>
#include        "../host/ray_type.h"
#include        "../host/ray_vec.h"
#include        "../host/ray_comm.h"
#include        "node_extern.h"

#define EPSIcmp         1e-05
#define MIN(A,B)        ((A) <  (B) ? (A) : (B))
#define MAX(A,B)        ((A) >= (B) ? (A) : (B))

Flt     tab_S[3];
Flt     tab_T[3];
static  Vec     slabs[3] = { {1.,0.,0.}, {0.,1.,0.}, {0.,0.,1.} };

InitSlabs (pp)
     Poly *pp;
{
  int i,j;
  Flt d_near, d_far;

  for (i=0; i<SLABS; i++)       {       /* For each slab */
    d_near = MAXfloat;
    d_far  = MINfloat;
    for (j=0; j<3; j++) {               /* For each vertex */
      Flt d = PROD_SCAL(pp->point[j], slabs[i]);

      d_near = MIN(d_near, d);
      d_far  = MAX(d_far , d);
    }
    pp->sl[i].near = d_near - EPSIcmp;
    pp->sl[i].far  = d_far  + EPSIcmp;
  }
}

InterRaySlabs ()
{
  int    i;
  Flt   val;

  for (i=0; i<3; i++)   {       /* For each slab */
    tab_S[i] = PROD_SCAL(slabs[i], myray->ori);
    val      = PROD_SCAL(slabs[i], myray->dir);
    tab_T[i] = (val == 0) ? MAXfloat : 1/val;
  }
}

InterPolySlabs (poly, near, far)
     Poly       *poly;
     Flt        *near, *far;
{
  Flt   t1, t2, t;
  int           i = 0;
```

```
  while (VRAI)  {
    t1 = (poly->sl[i].near - tab_S[i]) * tab_T[i];
    t2 = (poly->sl[i].far  - tab_S[i]) * tab_T[i];

    if (t1>t2)  {
      t=t1; t1=t2; t2=t;
    }
    *near = MAX(*near, t1);
    *far  = MIN(*far , t2);

    if (*near > *far)   return (FAUX);
    if (++i==3)         return (VRAI);
  }
}
```



```
/*_____________________________________________________________________

  File          : node_extern.h
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

extern  Flt             x_min, x_max;
extern  Flt             y_min, y_max;
extern  Flt             z_min, z_max;

extern  Rayon           *myray;
extern  short           IDpho;

extern  short           traiter;
extern  Lumiere         t_lum[];
extern  Photo           t_pho[];

extern  Color           fond;
extern  Color           ambiant;
extern  Vue             vue;
extern  short           profondeur;
extern  long            n_pol;
extern  short           n_lum;
extern  long            rayID;

extern  short           xres;
extern  short           yres;
extern  short           bx1,bx2,by1,by2;
extern  int             seq;
extern  short           IDpho;

extern  short           vite;
extern  short           alias;
extern  short           zoom;
extern  Flt             seuil;

extern  long            *seads;
extern  int             ind1, ind2;
extern  short           nb_cell_x, nb_cell_y, nb_cell_z;
extern  Flt             dx_vox, dy_vox, dz_vox;

extern  long            Me, num_nodes, node_dim;
```

```
%{
/*
 *                  ray_nff.y
 *
 * This file is an adaptation of the corresponding file from MTV software.
 * Thus the contribution of Mark VandeWettering must be mentioned.
 *
 * authors:  Mark VANDERWETTERING - Didier BADOUEL
 */
#include <stdio.h>
#include <math.h>
#include "ray_type.h"
#include "ray_vec.h"
#include "ray_comm.h"
#include "host_extern.h"

#define ABS(X)          (((X) < 0) ? -(X) : (X))
#define PI              (3.14159265358979323844)
#define degtorad(x)     (((Flt)(x))*PI/180.0)

extern  char            yytext[];
extern  FILE            * yyin;
extern  int             yylinecount;
extern  Vue             vue;
extern  short           xres;
extern  short           yres;

int   identpho;
long  poly;
Flt   *pp, *np;
%}

%token VIEWPOINT FROM AT UP ANGLE HITHER RESOLUTION LIGHT
%token BACKGROUND SURFACE NUMBER GRID LIST POLYGON PATCH NUM

%union {
        Vec     vec ;
        Vec *   vecl ;
        Flt     flt ;
        Flt     yyfloat; /* Jean-Luc CORRE       */
        int     yyint;   /* Jean-Luc CORRE       */
        char    *yystr;  /* Jean-Luc CORRE       */
} ;
%token  <yyint>         INT
%token  <yyfloat>       FLOAT
%token  <yystr>         STRING

%type <vec>     point primcolor vecteur sommet
%type <flt>     num

%%

scene:
  camera elementlist
{
  int l ;
  Flt coeff = (Flt)(sqrt((double)n_lum)/(double)n_lum) ;
  for (l = 0 ; l < n_lum ; l++) {
    t_lum[l].coeff = coeff;
  }
  ambiant[0] = coeff;
  ambiant[1] = coeff;
  ambiant[2] = coeff;
}

elementlist:
  elementlist element
  | ;

element:
  light
  | background
  | surface
  | list
  | grid
  | polygon
  | ppatch ;

camera:
  VIEWPOINT                     /* $1      */
  FROM point                    /* $2-$3   */
  AT point                      /* $4-$5   */
  UP point                      /* $6-$7   */
  ANGLE num                     /* $8-$9   */
  HITHER num                    /* $10-$11 */
  RESOLUTION num num            /* $12-$14 */
  NUMBER num                    /* $15-$16 */
{
  bcopy((char *)$3, (char *)vue.from, sizeof(Point));
  bcopy((char *)$5, (char *)vue.at, sizeof(Point));
  bcopy((char *)$7, (char *)vue.up, sizeof(Vec));
  vue.angle = degtorad($9/2.0);
  vue.dist = $11;
  xres = (short) $13;
  yres = (short) $14;
  n_pol = $16;
}

light:
  LIGHT point
{
  if (n_lum >= MAXLUM) {
    fprintf(stderr,"too much light sources\n");
  } else {
    bcopy((char *)$2, (char *)t_lum[n_lum].pos, sizeof(Point));
    t_lum[n_lum].type = LUMI;
    /* fill in brightness of the light, after we
     * know the number of lights sources in the scene
     */
    n_lum++ ;
  }
}

background:
  BACKGROUND primcolor
{
  bcopy((char *)$2, (char *)fond, sizeof(Color)) ;
}

surface:
  SURFACE primcolor num num num num num
{
  if (n_pho >= MAXPHO) {
    fprintf(stderr,"too photometric descriptions\n");
  } else {
```

```
    Photo       photo;

    bcopy((char *)$2, (char *)photo.color, sizeof(Color));
    photo.kd = $3 ;
    photo.ks = $4 ;
    photo.phong = $5 ;
    photo.kt = $6 ;
    photo.ior = $7 ;
    for (identpho=0; identpho<n_pho; identpho++) {
      if (memcmp((char*)&photo,(char*)&(t_pho[identpho]),sizeof(Photo))==0)
        break;
    }
    if (identpho==n_pho) {
      bcopy((char*)&photo,(char*)&(t_pho[n_pho]),sizeof(Photo));
      n_pho++;
    }
  }
}

list:
  LIST bounding

grid:
  GRID bounding
{
  InitSeads ();
}

polygon:
  POLYGON num
{
  t_pol[ip].poly = poly;
  t_pol[ip].rayID = 0;
  t_pol[ip].f_plan = 1;
  t_pol[ip].photo = identpho;
  n_som = 0;
  pp = (Flt*)t_pol[ip].point;
  np = (Flt*)t_pol[ip].normal;
}

pointlist
{
  RangerEltSeads (ip);
  ip++;poly++;
  if ((ip==poly_page)||(poly==n_pol)) {
    SendPage((char *)t_pol);
    ip = 0;
  }
}

ppatch:
  PATCH num
{
  t_pol[ip].poly = poly;
  t_pol[ip].rayID = 0;
  t_pol[ip].f_plan = 0;
  t_pol[ip].photo = identpho;
  n_som = 0;
  pp = (Flt*)t_pol[ip].point;
  np = (Flt*)t_pol[ip].normal;
}
sommetlist
{
  RangerEltSeads (ip);
  ip++;poly++;
  if ((ip==poly_page)||(poly==n_pol)) {
    SendPage((char *)t_pol);
    ip = 0;
  }
}
bounding:
  num num num num num num
{
  x_min = $1;
  x_max = $2;
  y_min = $3;
  y_max = $4;
  z_min = $5;
  z_max = $6;
}
primcolor:
  num num num
{
  $$[0] = $1 ;
  $$[1] = $2 ;
  $$[2] = $3 ;
}

point:
  num num num
{
  $$[0] = $1 ;
  $$[1] = $2 ;
  $$[2] = $3 ;
}

vecteur:
  num num num
{
  if (n_som < 4) {
    *np++ = $1;
    *np++ = $2;
    *np++ = $3;
  }
}

sommet:
  num num num
{
  if (n_som < 4) {
    *pp++ = $1;
    *pp++ = $2;
    *pp++ = $3;
  }
}

pointlist:
  pointlist sommet
{  n_som++; }
| ;

sommetlist:
  sommetlist sommet vecteur
{ n_som++; }
| ;
```

```
num
   : FLOAT
{ $$ = $1; }
| INT
{ $$ = (Flt) $1; };
%%

#include        "my.h"
#include        "lex.h"

fatal (str)
     char    *str;
{
  fprintf (stderr, "%s: %d: error %s\n", infile, yylineno, str);
  EXIT (1);
}

warning (str)
     char    *str;
{
  fprintf (stderr, "%s: %d: warning %s\n", infile, yylineno, str);
  EXIT (1);
}

yyerror(str)
{
  fprintf(stderr, "error at line %d\n", yylineno) ;
  EXIT(-1) ;
}

yywrap()
{
  return 1 ;
}

ReadSceneFile(str)
     char *str ;
{
  /*
   * KEYWORD_TBL : Table for key words and tokens for the grammar.
   */
  static  Keyword keyword_tbl[] = {
    "v",            VIEWPOINT,      TRUE,
    "viewpoint",    VIEWPOINT,      TRUE,
    "from",         FROM,           TRUE,
    "at",           AT,             TRUE,
    "up",           UP,             TRUE,
    "angle",        ANGLE,          TRUE,
    "hither",       HITHER,         TRUE,
    "resolution",   RESOLUTION,     TRUE,
    "l",            LIGHT,          TRUE,
    "light",        LIGHT,          TRUE,
    "b",            BACKGROUND,     TRUE,
    "background",   BACKGROUND,     TRUE,
    "g",            GRID,           TRUE,
    "grid",         GRID,           TRUE,
    "list",         LIST,           TRUE,
    "n",            NUMBER,         TRUE,
    "number",       NUMBER,         TRUE,
    "f",            SURFACE,        TRUE,
    "surface",      SURFACE,        TRUE,
    "p",            POLYGON,        TRUE,
    "polygon",      POLYGON,        TRUE,
    "pp",           PATCH,          TRUE,
    "patch",        PATCH,          TRUE,
    NULL,           0,              TRUE
    };
  open_lex ();
  open_keyword (keyword_tbl);
  strncpy (infile, str, NAMESIZE);
  infile[NAMESIZE-1] = '\0';
  if (!strcmp(str,"stdin")) {
    lexin = stdin ;
    fprintf(stderr, "\t\"stdin\"\n") ;
  }
  else {
    if ((lexin = fopen(str, "r")) == NULL) {
      fprintf(stderr, "cannot open %s\n", str) ;
      EXIT(-1) ;
    }
    fprintf(stderr, "\t\"%s\"\n", str) ;
  }
  poly = 0;
  lexout = stderr;
  open_lexio ();
  if (yyparse() == 1) {
    fprintf(stderr, "invalid input specification\n");
    EXIT(-1) ;
  }
  close_lex ();
  close_keyword ();
  close_lexio ();
  fprintf(stderr,
          "\t%d polygons\t\t\n\t%d light sources\n\t%d surface types\n",
          poly, n_lum, n_pho);
}
```

```
/*_____________________________________________________________________

        File        : ray_vect.h
        Author      : Didier BADOUEL
                      (c) IRISA / University of RennesI  - 1989
_____________________________________________________________________*/

#define PROD_SCAL(A,B)   ((A[0])*(B[0])+(A[1])*(B[1])+(A[2])*(B[2]))
#define TAILLE_VEC(A)    ((Flt)sqrt((double)PROD_SCAL(A,A)))
#define VECTEUR(A,B,C)   {      (C[0])=(B[0]) - (A[0]);\
                                (C[1])=(B[1]) - (A[1]);\
                                (C[2])=(B[2]) - (A[2]);\
                         }
#define PROD_VEC(A,B,C)  {      (C[0])=(A[1])*(B[2])-(A[2])*(B[1]);\
                                (C[1])=(A[2])*(B[0])-(A[0])*(B[2]);\
                                (C[2])=(A[0])*(B[1])-(A[1])*(B[0]);\
                         }
#define NORM_VEC(A)      {Flt   len = TAILLE_VEC(A);\
                                (A[0]) /= len;\
                                (A[1]) /= len;\
                                (A[2]) /= len;\
                         }
#define ADD_VEC(A,B,C)   {      (C[0])=(A[0])+(B[0]);\
                                (C[1])=(A[1])+(B[1]);\
                                (C[2])=(A[2])+(B[2]);\
                         }
#define SUB_VEC(A,B,C)   {      (C[0])=(A[0])-(B[0]);\
                                (C[1])=(A[1])-(B[1]);\
                                (C[2])=(A[2])-(B[2]);\
                         }
#define ALPHA_VEC(A,B,C){       (C[0])=(A)*(B[0]);\
                                (C[1])=(A)*(B[1]);\
                                (C[2])=(A)*(B[2]);\
                         }
#define ALPHA_COMB_VEC(N,A,M,B,C)\
                         {      (C[0])=(N)*(A[0])+(M)*(B[0]);\
                                (C[1])=(N)*(A[1])+(M)*(B[1]);\
                                (C[2])=(N)*(A[2])+(M)*(B[2]);\
                         }
#define COMB_VEC(A,B,C)  {      (C[0])=(A[0])*(B[0]);\
                                (C[1])=(A[1])*(B[1]);\
                                (C[2])=(A[2])*(B[2]);\
                         }
#define INV_VEC(A)       {      (A[0])= -(A[0]);\
                                (A[1])= -(A[1]);\
                                (A[2])= -(A[2]);\
                         }
```



```
/*_____________________________________________________________________

  File          : ray_page.h
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
_____________________________________________________________________*/

#define PAGESIZE        1280
#define PAGENUMB        2550
#define OBJNUMB         5
#define WAY             2
#define ENTRY           (int)(PAGENUMB/WAY)

typedef char Page[PAGESIZE];

typedef struct {
        long    page;   /* first page of the object     */
        int     numb;   /* number of objects per page   */
        int     size;   /* size of an object            */
        int     dim;    /* log2(size)                   */
} Object;

typedef Object  Objdesc[OBJNUMB];

typedef Page    Mem[PAGENUMB];

typedef struct {
  long index;
#if (WAY > 1)
  long date;
#endif
  long entree;
} Elt;

typedef Elt     Cache[WAY];
```

```
/*_____________________________________________________________________

  File          : ray_comm.h
  Author        : Didier BADOUEL
                  (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________*/

#define NL              4
#define NB              50
#define PID             100
#define HOST_PID        100
#define NODE_PID        100

#define ALL_NODES       -1
#define ALL_PIDS        -1

#define short_TYPE       1
#define int_TYPE         2
#define Flt_TYPE         3
#define long_TYPE        4

#define DEM_TYPE         6
#define PAGE_TYPE        7
#define OBJ_TYPE         8
#define ACK_TYPE         9
#define DEM_PACK        11
#define P_INIT_TYPE     50
#define P_LUM_TYPE      60
#define P_POL_TYPE      70
#define P_PHO_TYPE      80
#define COUL_TYPE       90
#define STATUS_TYPE    100
#define SEADS_TYPE     120
#define PACK_TYPE      130
#define GO_TYPE        200

typedef struct {
        int type;
        long n;
} MSG_dem;

typedef struct  {
        short nb;
        short i[NB];
        short j[NB];
        unsigned char buf[NB][NL*NL*3];
} MSG_coul;

typedef struct  {
        short   xres;
        short   yres;
        short   vite;
        short   n_lum;
        short   n_pho;
        long    n_pol;
        Color   fond;
        Color   ambiant;
        short   profondeur;
        Vue     vue;
} MSG_p_init;

typedef struct {
        long    node;
        short   x;
        short   y;
} MSG_pack;
```

```
/*_____________________________________________________________________

  Filer          : ray_type.h
  Author         : Didier BADOUEL
                   (c) IRISA / University of RennesI  - 1989
  _____________________________________________________________________
  An object never serves the same function as its image or its name.
                                                        Rene Magritte.    */

#define bcopy(A,B,N)    (void)memcpy(B,A,N)
#define bzero(A,N)      (void)memset(A,0,N)
#define EXIT(N)         { killcube(-1, -1); exit(N); }
#define VRAI            1
#define FAUX            0
#define EPSILON         1e-05
#define MAXshort        32767                   /*  2^15 - 1    */
#define MAXlong         2147483647              /*  2^31 - 1    */
#define MAXfloat        ((Flt) 1.70141173319264429e+38)
#define MINfloat        ((Flt)-1.70141173319264429e+38)
#define MAXLUM          10
#define MAXPHO          20
#define MAXVOX          32768
#define NB_ELT_LIST     8                       /* ! must be 2^n ! */
#define SEADS           70
#define COEFF           10
/*
 * types definition.
 */
typedef float Flt;
typedef Flt Matrix[4][4];
typedef Flt Vec[3];
typedef Flt Vec4[4];
typedef Vec4 Equ;
typedef Vec Point;
typedef Vec Color;
typedef long Voxlist[NB_ELT_LIST];

#define PRIM            0
#define SECON           1
#define LUM             2

typedef struct  {
        long            type;
        long            rayID;
        Point           ori;
        Vec             dir;
        short           photo;
        short           niv;
        short           ipix;
        short           jpix;
        Flt             maxlum;
        Flt             ombre;
        Color           coeff;
        Flt             t;
        long            poly;
        Point           point;
        Vec             normal;
} Rayon;

#define LUMI            0
#define SPOT            1

typedef struct  {
        char            type;
        Point           pos;
        Vec             dir;
        Flt             coeff;
} Lumiere;

typedef struct  {
        Color           color;
        Flt             kd;
        Flt             ks;
        short           phong;
        Flt             kt;
        Flt             ior;
} Photo;

#define SLABS 3
typedef struct  {
        Flt     near,far;
} Slab;

typedef struct  {               /* ! sizeof(Poly) must be 2^n ! */
        long    poly;
        long    rayID;
        short   f_plan;
        short   photo;
        Equ     plan;
        short   i1, i2;
        Slab    sl[SLABS];
        Point   point[3];
        Vec     normal[3];
} Poly;

typedef struct t_vue {
        Vec     from;
        Vec     at;
        Vec     up;
        Flt     angle;
        Flt     dist;
} Vue;
```

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 506 **VM_pRAY, AN EFFICIENT RAY TRACING ALGORITHM ON DISTRIBUTED MEMORY PARALLEL COMPUTER**
Didier BADOUEL, Thierry PRIOL
Janvier 1990, 50 Pages.

PI 507 **ON THE REGULAR STRUCTURE OF PREFIX REWRITINGS**
Didier CAUCAL
Janvier 1990, 32 Pages.

PI 508 **RAY TRACING ON DISTRIBUTED MEMORY PARALLEL COMPUTERS: STRATEGIES FOR DISTRIBUTING COMPUTATIONS AND DATA.**
Didier BADOUEL, Kadi BOUATOUCH, Thierry PRIOL
Janvier 1990, 16 Pages.

PI 509 **STABILITY ANALYSIS AND IMPROVEMENT OF THE BLOCK GRAM-SCHMIDT ALGORITHM**
William JALBY, Bernard PHILIPPE
Janvier 1990, 24 Pages.

PI 510 **TESTING FOR THE UNBOUNDEDNESS OF FIFO CHANNELS IN PROGRAMS**
Thierry JERON
Janvier 1990, 30 Pages.

PI 511 **AUTOMATIC ANIMATION CONTROL OF PHYSICAL SYSTEMS**
Georges DUMONT, Bruno ARNALDI, Gérard HEGRON
Janvier 1990, 22 Pages.

Imprimé en France
par
. l'Institut National de Recherche en Informatique et en Automatique .